RESUMO PEP 8

FERNANDO QUINTILIANO FARIA JUNIOR

Introdução

- As diretrizes fornecidas pelo pep 8 têm o objetivo de melhorar a legibilidade do código e torná-lo consistente em todo o espectro do código Python
- Às vezes, as recomendações do guia de estilo simplesmente não são aplicáveis

Indentação

- Use 4 espaços por nível de indentação
- As linhas de continuação devem alinhar os elementos quebrados verticalmente usando a junção de linha implícita do Python dentro de parênteses, colchetes e chaves, ou usando uma indentação suspensa

```
# Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

- Ao usar uma indentação suspensa, deve-se considerar o seguinte: não deve haver argumentos na primeira linha e a indentação adicional deve ser usada para distinguir claramente como uma linha de continuação
- A regra dos 4 espaços é opcional para linhas de continuação

- O fechamento de colchetes/parênteses/chaves em construções multilinha pode ser alinhado sob o primeiro caractere não de espaço em branco da última linha da lista, como em:

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

or it may be lined up under the first character of the line that starts the multiline construct, as in:

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

Tabs ou Espaços?

- Espaços!!!!!!!
- Para blocos longos de texto com menos restrições estruturais (docstrings ou comentários), o comprimento da linha deve ser limitado a 72 caracteres
- A maneira preferida de quebrar linhas longas é usando a continuação de linha implícita do Python dentro de parênteses, colchetes e chaves. Linhas longas podem ser quebradas em várias linhas envolvendo expressões em parênteses. Estes devem ser usados preferencialmente ao usar uma barra invertida para continuação de linha

  Linhas em branco

  - Envolver definições de funções e classes de nível superior com duas linhas em branco
  - Use linhas em branco em funções, moderadamente, para indicar seções lógicas

  Codificação do arquivo

  - Todos os identificadores na biblioteca padrão do Python DEVEM usar identificadores somente ASCII e DEVEM usar palavras em inglês sempre que possível (em muitos casos, são usadas abreviações e termos técnicos que não são em inglês)

  Importações

  - Geralmente devem estar em linhas separadas

- As importações devem ser colocadas sempre no topo do arquivo, logo após quaisquer comentários e docstrings do módulo, e antes das variáveis globais e constantes do módulo
- As importações devem ser agrupadas na seguinte ordem: importações da biblioteca padrão, importações de terceiros e importações específicas da aplicação/biblioteca local
- O código da biblioteca padrão deve evitar layouts complexos de pacotes e sempre usar importações absolutas

Espaços em Branco em Expressões e Declarações

- Evite espaços em branco desnecessários nas seguintes situações:
    - Imediatamente dentro de parênteses, colchetes ou chaves
    - Entre uma vírgula final e um parêntese de fechamento seguinte
    - Imediatamente antes de uma vírgula, ponto e vírgula ou dois-pontos
    - No entanto, em um fatiamento, os dois-pontos atuam como um operador binário e devem ter quantidades iguais de espaço em ambos os lados
    - Evite espaços em branco no final de linhas. Como geralmente são invisíveis, podem ser confusos
    - Sempre coloque um espaço de cada lado desses operadores binários

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

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

Comentários

- Comentários que contradizem o código são piores do que nenhum comentário. Sempre dê prioridade a manter os comentários atualizados quando o código mudar
- Os comentários devem ser frases completas

- Programadores Python de países não anglófonos: escrevam seus comentários em inglês, a menos que tenham 120% de certeza de que o código nunca será lido por pessoas que não falam seu idioma.
- Certifique-se de que seus comentários sejam claros e facilmente compreensíveis para outros falantes do idioma em que você está escrevendo

Convenções de nomenclatura

- Nunca use os caracteres 'l' (letra minúscula ele), 'O' (letra maiúscula ó) ou 'I' (letra maiúscula í) como nomes de variáveis de um único caractere
    - Em algumas fontes, esses caracteres são indistinguíveis dos números um e zero. Quando tentado a usar 'l', use 'L' em vez disso
- Módulos devem ter nomes curtos, todos em minúsculas. Underscores podem ser usados no nome do módulo se melhorarem a legibilidade. Pacotes Python também devem ter nomes curtos, todos em minúsculas, embora o uso de underscores seja desencorajado
- Nomes de variáveis de tipo introduzidos no PEP 484 devem normalmente usar CapWords, preferindo nomes curtos: T, AnyStr, Num.
- Recomenda-se adicionar os sufixos _co ou _contra às variáveis usadas para declarar comportamento covariante ou contravariante, respectivamente
- Como exceções devem ser classes, a convenção de nomenclatura de classes se aplica aqui. No entanto, você deve usar o sufixo "Error" nos nomes das suas exceções (se a exceção realmente for um erro)
- Os nomes de funções devem ser em minúsculas, com palavras separadas por underscores conforme necessário para melhorar a legibilidade
- Nomes de variáveis seguem a mesma convenção dos nomes de funções

- Sempre use self para o primeiro argumento dos métodos de instância
- Sempre use cls para o primeiro argumento dos métodos de classe
- Geralmente, underscores duplos iniciais devem ser usados apenas para evitar conflitos de nomes com atributos em classes projetadas para serem subclassificadas

Herança

- Sempre decida se os métodos e variáveis de instância de uma classe (coletivamente: “atributos”) devem ser públicos ou não públicos. Em caso de dúvida, escolha não públicos; é mais fácil torná-los públicos mais tarde do que tornar um atributo público não público
- Atributos públicos não devem ter underscores no início
- Se o nome do seu atributo público colidir com uma palavra reservada, acrescente um único underscore ao final do nome do seu atributo

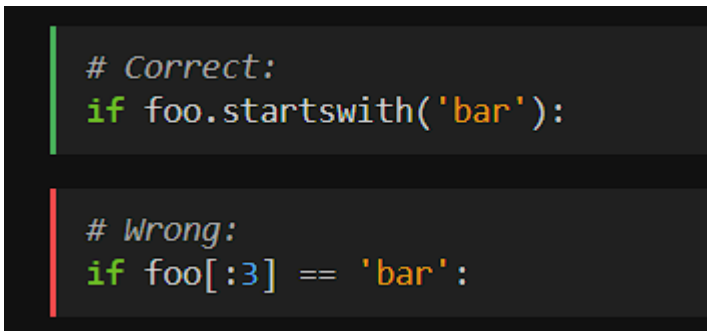
Recomendações de programação

- O código deve ser escrito de forma a não desfavorecer outras implementações do Python
- Use o operador is not em vez de not ... is. Embora ambas as expressões sejam funcionalmente idênticas, a primeira é mais legível e preferida
- Quando pegar exceções, mencione exceções específicas sempre que possível em vez de usar uma cláusula **except** simples
- Seja consistente nas declarações de retorno.
- Todas as declarações de retorno em uma função devem retornar uma expressão ou nenhuma delas deve retornar.
- Se alguma declaração de retorno retornar uma expressão, todas as declarações de retorno em que nenhum valor é retornado devem declarar explicitamente isso como **return None**, e uma declaração de retorno explícita deve estar presente no final da função (se alcançável)

```python
# Correct:

def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None

def bar(x):
    if x < 0:
        return None
    return math.sqrt(x)
```

```python
# Wrong:

def foo(x):
    if x >= 0:
        return math.sqrt(x)

def bar(x):
    if x < 0:
        return
    return math.sqrt(x)
```

- Use **".startswith()** e **".endswith()** em vez de fatiar strings para verificar prefixos ou sufixos. **startswith()** e **endswith()** são mais limpas e menos propensas a erros:



- Não compare valores booleanos com True ou False usando ==
- Anotações para variáveis de nível de módulo, variáveis de classe e instância e variáveis locais devem ter um único espaço após os dois pontos

```python
# Correct:

code: int

class Point:
    coords: Tuple[int, int]
    label: str = '<unknown>'
```

```python
# Wrong:

code:int  # No space after colon
code : int  # Space before colon

class Test:
    result: int=0  # No spaces around equality sign
```