Resumo: PEP 8 - Style Guide for Python Code

Larissa Broggio Raymundo

Introdução:

O PEP 8 é um guia dinâmico para estilo de códigos em Python que tem como objetivo a consistência e melhor legibilidade dos códigos. Apesar disso, há exceções – como quando aplicar a diretriz tornaria o código menos legível ou incompatível com versões anteriores.

Organização do código:

- **Indentação:**
  - Usar 4 espaços por nível de indentação;
  - Dividir linha longa de código em várias linhas (mais organizado e legível) usando parênteses, colchetes ou chaves;
  - Recuar a linha para dar continuação (evitar linhas longas);
  - Quando se tem condicional longa pode ser feito sem indentação extra, adicionando um comentário ou uma indentação extra;
  - O fechamento da chave/colchete/parênteses em várias linhas pode ser alinhado sob o primeiro caractere não vazio da última linha da lista (com a indentação) ou sob o primeiro caractere da linha que inicia a construção de várias linhas (sem a indentação).

- **Tabs ou espaços:**
  - A preferência é em espaços;
  - Pode utilizar tab somente em casos de códigos já indentados com tab para não ficar diferente;
  - Não misturar tabs e espaços.

- **Tamanho máximo da linha:**
  - Limitar a 79 caracteres;
  - Longos blocos de texto sem restrições estruturais (comentários e docstrings): limitar a 72 caracteres;
  - Limitar o tamanho da janela do editor para poder abrir janelas lado a lado;
  - Limites evitam quebra automática;
  - Equipes podem aumentar o limite para 99 caracteres, mas comentários e docstrings ainda devem ser mantidos em até 72;
  - De preferência, quebrar linhas longas com continuação implícita (dentro de parênteses, colchetes e chaves);
  - Barras invertidas (\) eram aceitáveis para quebrar linhas antes do Python 3.1.

- **Quebrar uma linha antes ou depois de um operador binário?**
  - No estilo tradicional, quebrava-se antes. Porém, isso pode desconectar visualmente o operador do operando correspondente;
  - É flexível, desde que siga o mesmo padrão no código todo;
  - O recomendado para novos códigos é utilizar a quebra antes.

- **Linhas em branco**
  - Usadas para separar visualmente definições de funções e indicar sessões lógicas;
  - Usar moderadamente;
  - Pode usar control-L (^L) como espaço em branco, porém alguns editores e visualizadores de código podem não reconhecer e mostrar outro caractere em seu lugar.

- **Codificação de arquivos fonte**
  - O código na distribuição principal do Python deve sempre usar UTF-8 (Formato de transformação Unicode de 8 bits, é um padrão de codificação de caracteres que pode representar todos os caracteres no conjunto de caracteres Unicode) e não deve ter uma declaração de codificação;
  - Codificações não UFT-8 podem ser usadas para fins de teste;
  - Todos os identificadores na biblioteca padrão do Python e projetos de código aberto devem usar apenas identificadores ASCII e palavras em inglês sempre que possível.

- **Importações**
  - Devem estar em linhas separadas;
  - São colocadas no topo do arquivo, logo após quaisquer comentários de módulo e docstrings;
  - Devem seguir a ordem: 1 - Importações da biblioteca padrão; 2- Importações de bibliotecas de terceiros relacionadas; 3- Importações específicas de aplicativos ou bibliotecas locais;
  - Colocar uma linha em branco entre cada grupo de importações;
  - Importações absolutas são recomendadas (especifica o caminho completo, como: import meu_pacote.meu_modulo). Alternativa: importações relativas explícitas (especifica o caminho em relação ao módulo, como: from . import meu_modulo);
  - Evitar layouts de pacotes complexos;
  - Para importar uma classe de um módulo que contém classes, escrever assim: from myclass import MyClass
    from foo.bar.yourclass import YourClass

- **Nomes do Dunder de nível de módulo**
  - São nomes em módulos Python que têm dois sublinhados iniciais e dois finais, como __all__, __author__ e __version__. Esses nomes são considerados especiais e geralmente são usados para fins específicos no módulo;
  - Colocar após a docstring do módulo, mas antes de qualquer instrução de importação, exceto para importações de __future__ ;

- **Citações de String**
  - Pode usar aspas simples ou duplas, mas deve manter a consistência até o final.

- **Espaços em expressões e declarações**
  - Evitar espaços extras em:
    - 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;
    - Imediatamente antes do parêntese aberto que inicia a lista de argumentos de uma chamada de função; Imediatamente antes do parêntese aberto que inicia uma indexação ou fatiamento;
    - Mais de um espaço ao redor de um operador de atribuição para alinhá-lo com outro.
  - Outras recomendações:
    - Evitar espaços em branco no final de qualquer lugar;
    - Colocar operadores binários com um único espaço de cada lado: atribuição (=), atribuição aumentada (+=, -= etc.), comparações (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not);
    - Se operadores com diferentes prioridades forem usados, pode adicionar espaços ao redor dos operadores com a(s) menor(es) prioridade(s);
    - Anotações de função devem seguir as regras normais para dois pontos e sempre ter espaços em torno da seta -> se presente;
    - Não usar espaços em torno do sinal de = quando usado para indicar um argumento de palavra-chave ou um valor padrão para um parâmetro de função não anotado;
    - Para combinar uma anotação de argumento com um valor padrão, usar espaços em torno do sinal de = ;
    - Não fazer declarações compostas (múltiplas declarações na mesma linha).

- **Quando usar vírgulas finais**
  - São opcionais, exceto quando são obrigatórias ao criar uma tupla de um elemento;
  - São úteis quando se espera que uma lista de valores, argumentos ou itens importados seja estendida ao longo do tempo. O padrão é colocar cada valor em uma linha separada, sempre adicionando uma vírgula final, e adicionar o parêntese/colchete/chave de fechamento na próxima linha (não colocar na mesma linha!).

- **Comentários**
  - Manter atualizado junto com o código para não contradizer;
  - Deve ser frase completa, clara e objetiva;
  - A primeira palavra deve ser capitalizada (#);
  - Em bloco: um ou mais parágrafos, com frases completas;
  - Usar um ou dois espaços após um ponto que termina a sentença em comentários de várias sentenças, exceto após a última sentença;
  - Em inglês;
  - Em linha: deve ser usado com parcimônia. Deve ser separado por dois espaços.

- **Strings de documentação (docstrings)**
  - Escrever docstrings para todos os módulos, funções, classes e métodos públicos;
  - Usados entre aspas triplas (“”“) e colocados no início da definição do módulo, função, classe ou método.

- **Convenções de nomenclatura**
  - Novos módulos e pacotes devem ser escritos de acordo com esses padrões, mas onde uma biblioteca existente tiver um estilo diferente, seguir a consistência interna;
  - Princípios de substituição: Nomes que são visíveis para o usuário devem seguir convenções que refletem o uso;
  - Estilos de nomenclatura: minúscula, maiúscula, com ou sem underscore, CapWords;
  - Formas especiais usando sublinhados iniciais ou finais para distinguir de palavras-chave do Python;
  - Evitar: “l”, “o” e “u” como nomes de variáveis de um único caractere;
  - Identificadores usados na biblioteca padrão devem ser compatíveis com ASCII (American Standard Code for Information Interchange);
  - Nomes de pacotes e módulos: devem ter nomes curtos em minúsculas, pode usar underscore;
  - Módulo C/C++ tem um sublinhado inicial;
  - Nomes de classes: devem seguir a convenção CapWords;
  - Nomes de variáveis de tipo: devem usar CapWords e serem curtos: T, AnyStr, Num. Recomenda-se adicionar sufixos _co ou _contra às variáveis usadas para declarar comportamento covariante ou contravariante;
  - Nomes de exceção: deve usar o sufixo “Error” em seus nomes de exceção (se a exceção realmente for um erro);
  - Nomes de variáveis globais: Módulos destinados a serem usados via from M import * devem usar o mecanismo all para evitar a exportação de globais;
  - Nomes de funções e variáveis devem ser em letras minúsculas;
  - Argumentos de função e método: sempre use self para o primeiro argumento em métodos de instância e cls para o primeiro argumento em métodos de classe;
  - Se o nome de um argumento de função entra em conflito com uma palavra-chave reservada: adicionar um único sublinhado final;
  - Nomes de métodos e variáveis de instância: letras minúsculas com palavras separadas por sublinhados conforme necessário para melhorar a legibilidade;
  - Constantes: definidas em nível de módulo e escritas em todas as letras maiúsculas com sublinhados separando palavras;
  - Atributos: Deixar não-publico - pode tornar público depois, mas ao contrário é mais difícil;
  - Atributos públicos são aqueles que você espera que clientes não relacionados de sua classe usem; Atributos públicos não devem ter sublinhados iniciais; Se o nome do atributo público colidir com uma palavra-chave reservada, adicionar um sublinhado simples no final do nome do atributo.

- **Recomendações**
  - Escrever de forma que não prejudique outras implementações dePython;
  - Usar is ou is.not ao invés de operadores de igualdade (== !=) para evitar confusões;
  - Lambda: usar declaração def ao invés de atribuir uma expressão lambda;
  - Derivar exceções de “Exception”;
  - Usar o encadeamento de exceções apropriadamente: `raise X from Y` deve ser usado para indicar uma substituição explícita;
  - Quando um recurso é local: a uma seção específica do código, use uma instrução “with” para garantir que ele seja limpo prontamente após o uso;
  - Usar métodos de string como “startswith()” e “endswith()” em vez de fatiar strings para verificar prefixos ou sufixos;
  - Comparações de tipo de objeto: sempre usar `isinstance()` em vez de comparar tipos diretamente;
  - Evitar escrever string que dependam de espaços em branco significativos no final;
  - Não comparar valores booleanos com “True” ou “False” usando “==”.

- **Anotações de função**
  - Fora da stdlib (“*standard library*”, biblioteca padrão do Python), experimentos dentro das regras do PEP 484 são incentivados;
  - Para quem deseja fazer um uso diferente das anotações de função, recomenda-se colocar um comentário # type: ignore, que diz aos verificadores de tipo para ignorarem todas as anotações.

- **Anotações de variáveis**
  - Semelhantes à de função;
  - Deve ter um único espaço após os dois pontos;
  - Não deve haver espaço antes dos dois pontos;
  - Se uma atribuição tiver um lado direito, o sinal de igualdade deve ter exatamente um espaço em ambos os lados.