

UNIVERSIDADE FEDERAL DE SÃO CARLOS

PEP 8 - O GUIA DE ESTILO PARA CÓDIGO PYTHON

UM RESUMO PARA INICIANTEs

JESUS DAVID SIERRA MARTINEZ

SÃO CARLOS, 2024

JESUS DAVID SIERRA MARTINEZ

PEP 8 - O GUIA DE ESTILO PARA CÓDIGO PYTHON
UM RESUMO PARA INICIANTES

Trabalho para a disciplina: Python para Biocientistas

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Professor Dr. Célio Dias Santos Júnior

São Carlos

2024

LISTA DE ILUSTRAÇÕES

Figura 1. Elementos alinhados para correta indentação.....	6
Figura 2: Opções de escrita quando o condicional da instrução é cumprido.....	7
Figura 3. Operador binário situado após a quebra.....	7
Figura 4. Módulos importados de forma organizada e evitando importações desnecessárias...8	
Figura 5. Ordem de <i>dunders</i> após <i>docstrings</i> e antes de instruções de importação.....	9
Figura 6. Declarações em linhas separadas melhoram a leitura.....	11
Figura 7. Vírgulas finais são usadas ao criar uma tupla.....	11
Figura 8. Múltiplas vírgulas finais podem ser usadas em diferentes linhas.....	11
Figura 9. Fechamento de <i>docstrings</i> multilinha.....	12

LISTA DE TABELAS

Tabela 1. Uso correto de espaço em expressões.....	9
---	---

SUMARIO

INTRODUÇÃO.....	5
1. DELINEAMENTO DO CÓDIGO.....	6
2. CITAÇÕES DE <i>STRINGS</i>	9
3. ESPAÇO EM BRANCO EM EXPRESSÕES E DECLARAÇÕES.....	9
4. OUTRAS RECOMENDAÇÕES.....	10
5. QUANDO USAR VÍRGULAS FINAIS?.....	11
6. COMENTÁRIOS.....	11
7. CONVENÇÕES DE NOMENCLATURA.....	12
8. RECOMENDAÇÕES DE PROGRAMAÇÃO.....	14
CONCLUSÃO.....	16
REFERÊNCIAS.....	17

INTRODUÇÃO

Python é uma linguagem de programação muito intuitiva e poderosa que tem conquistado um lugar especial no desenvolvimento de programas informáticos e processamento de uma ampla variedade de dados. Python é caracterizada pela sua sintaxe concisa e clara, que dá prioridade a legibilidade do código sobre a velocidade ou expressividade, além disso, os recursos poderosos de sua biblioteca padrão, módulos e frameworks desenvolvidos por terceiros fazem com que sua utilidade seja intensamente dinâmica.

Aprender a programar com Python contribui na pesquisa biológica, já que é possível analisar grandes conjuntos de dados biológicos, aplicar técnicas de Machine Learning para extrair informações valiosas dos dados biológicos, realizar análises preditivas, modelagem e simulação, visualizar descobertas de maneira clara e eficiente, entre outras, o que abre portas para novas descobertas e avanços na área científica.

Com tudo, como toda linguagem, as criações em Python, sejam elas *scripts*, pacotes ou programas, precisam ser bem escritos, com clareza e eficiência para serem eficientemente interpretados, executados e difundidos na comunidade de usuários. É por isso que foi criado um consenso de convenções de código para Python chamado PEP-8 que facilita e harmoniza a a escritura e leitura desta linguagem.

No presente trabalho se apresenta um resumo das diretrizes PEP-8, abordando de forma simples sus aspectos principais com o intuito de fortalecer o aprendizado da programação em Python e facilitar a iniciação de outros biocientistas na área da bioinformática.

1. DELINEAMENTO DO CÓDIGO

Indentação

Use 4 espaços por nível de recuo:

Linhas de continuação devem alinhar elementos envolvidos verticalmente usando a junção implícita de linhas do Python dentro de parênteses, colchetes e chaves, ou utilizando um recuo suspenso. Ao utilizar um recuo suspenso, o seguinte deve ser considerado: não deve haver argumentos na primeira linha e um recuo adicional deve ser usado para claramente distingui-se como uma linha de continuação (**Figura 1**).

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

# More indentation included to distinguish this 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)
```

Figura 1. Elementos alinhados para correta indentação.

Quando a parte condicional de uma instrução é longa:

Quando for necessário que a instrução seja escrita em várias linhas, opções aceitáveis nessa situação são apresentadas na **Figura 2**:

```

# No extra indentation.
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

# Add a comment, which will provide some distinction in editors
# supporting syntax highlighting.
if (this_is_one_thing and
    that_is_another_thing):
    # Since both conditions are true, we can frobnicate.
    do_something()

# Add some extra indentation on the conditional continuation line.
if (this_is_one_thing
    and that_is_another_thing):
    do_something()

```

Figura 2: Opções de escrita quando o condicional da instrução é cumprido.

Tabulações ou espaços?

Espaços são o método de indentação preferido. As tabulações devem ser usadas apenas para permanecer consistente com o código que já está recuado com tabulações.

Comprimento máximo da linha

Limite as linhas de código a 79 caracteres para facilitar a leitura e a manutenção. Para blocos de texto longos e fluidos com menos restrições estruturais (documentos ou comentários), o comprimento da linha deve ser limitado a 72 caracteres.

Uma linha deve quebrar antes ou depois de um operador binário?

O recomendado é quebrar antes dos operadores binários, segue o exemplo na **Figura 3**

```

# Yes: easy to match operators with operands
income = (gross_wages
    + taxable_interest
    + (dividends - qualified_dividends)
    - ira_deduction
    - student_loan_interest)

```

Figura 3. Operador binário situado após a quebra.

Linhas em branco

Coloque as definições de função e classe de nível superior com duas linhas em branco. As definições de métodos dentro de uma classe são cercadas por uma única linha em branco. Use linhas em branco nas funções, com moderação, para indicar seções lógicas.

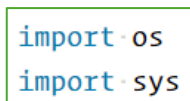
Codificação do arquivo de origem

Todos os identificadores na biblioteca padrão Python devem usar identificadores somente ASCII e devem usar palavras em inglês sempre que possível (em muitos casos, abreviações e informações técnicas são usados termos que não são ingleses). Além disso, literais de string e comentários também devem estar em ASCII. As únicas exceções são (a) casos de teste que testam os recursos não-ASCII e (b) nomes de autores. Autores cujos nomes não sejam baseados no alfabeto latino devem fornecer uma transliteração latina de seus nomes.

Importações

As importações são sempre colocadas no topo do arquivo, logo após quaisquer comentários e docstrings do módulo, e antes dos globais e constantes do módulo. As importações geralmente devem ser feitas em linhas separadas (**Figura 4**). A ordem sugerida das importações é a seguinte:

1. importações de biblioteca padrão
2. importações de terceiros relacionadas
3. importações específicas de aplicativos/bibliotecas locais



```
import os
import sys
```

Figura 4. Módulos importados de forma organizada e evitando importações desnecessárias.

Nomes “dunder” em nível de módulo

"Dunders" no nível do módulo (ou seja, nomes com dois sublinhados iniciais e dois finais), como `__all__`, `__author__`, `__version__`, etc. devem ser colocados após a *docstring* do módulo, mas antes de qualquer instrução de importação, exceto de importações `__future__`. Python determina que as importações futuras devem aparecer no módulo antes de qualquer outro código, exceto docstrings (**Figura 5**).

```

"""This is the example module.

This module does stuff.
"""

from __future__ import barry_as_FLUFL

__all__ = ['a', 'b', 'c']
__version__ = '0.1'
__author__ = 'Cardinal Biggles'

import os
import sys

```

Figura 5. Ordem de *dunders* após *docstrings* e antes de instruções de importação.

2. CITAÇÕES DE *STRINGS*

Em Python, strings entre aspas simples e strings entre aspas duplas são iguais. Escolha uma regra e cumpra-a. Quando uma string contém caracteres de aspas simples ou duplas, use a outra para evitar barras invertidas na string. Melhora a legibilidade.

3. ESPAÇO EM BRANCO EM EXPRESSÕES E DECLARAÇÕES

Pet Peeves

Evite espaços em branco desnecessários **Tabela (1)**.

Tabela 1. Uso correto de espaço em expressões.

Situação	Correto	Incorreto
<ul style="list-style-type: none"> Imediatamente dentro de parênteses, colchetes ou chaves 	<code>spam(ham[1], {eggs: 2})</code>	<code>spam(ham[1], { eggs: 2 })</code>
<ul style="list-style-type: none"> Entre uma vírgula final e um parêntese de fechamento subsequente 	<code>foo = (0,)</code>	<code>bar = (0,)</code>
<ul style="list-style-type: none"> Imediatamente antes de uma vírgula, ponto e vírgula ou dois pontos 	<code>if x == 4: print x, y; x, y</code>	<code>if x == 4: print x , y ; x , y</code>

- Imediatamente antes do parêntese de abertura que inicia a lista de argumentos de uma chamada de função
- Imediatamente antes do parêntese de abertura que inicia uma indexação ou *slicing*
- Mais de um espaço ao redor de um operador de atribuição (ou outro) para alinhá-lo com outro

`spam(1)`

`spam · (1)`

`dct['key'] := lst[index]`

`dct · ['key'] := lst · [index]`

`x := 1`

`y := 2`

`long_variable := 3`

`x ············ := 1`

`y ············ := 2`

`long_variable := 3`

Fonte: Rossum, G., Warsaw B. (2001).

4. OUTRAS RECOMENDAÇÕES

- Evite deixar espaços em branco em qualquer lugar.
- Sempre colocar esses operadores binários com um único espaço em cada lado: atribuição (=), atribuição aumentada (+=, -= etc.), comparações (==, <, >, !=, <=>, <=, >=, em, não em, é, não é), Booleanos (e, ou, não).
- Se forem usados operadores com prioridades diferentes, considere adicionar espaços em branco ao redor dos operadores com prioridade(s) mais baixa(s). Use seu próprio julgamento; entretanto, nunca use mais de um espaço e sempre tenha a mesma quantidade de espaços em branco em ambos os lados de um operador binário.
- Não use espaços ao redor do sinal = quando usado para indicar um argumento de palavra-chave ou um valor de parâmetro padrão.
- As anotações de função devem usar as regras normais para dois pontos e sempre ter espaços ao redor da seta ->, se presente.
- Declarações compostas (múltiplas declarações na mesma linha) são geralmente desencorajadas (**Figura 6**).

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

Figura 6. Declarações em linhas separadas melhoram a leitura.

- Embora às vezes seja correto colocar um if/for/while com um corpo pequeno na mesma linha, nunca fazer isso para instruções com múltiplas cláusulas. O recomendado é escrever as cláusulas em linhas separadas.

5. QUANDO USAR VÍRGULAS FINAIS?

As vírgulas finais são obrigatórias ao criar uma tupla de um elemento (**Figura 7**). Quando um sistema de controle de versão é usado, quando se espera que uma lista de valores, argumentos ou itens importados seja estendida ao longo do tempo, múltiplas virgulas finais podem ser usadas, sempre que cada elemento esteja em uma linha por si só, sempre adicionando uma vírgula final, e adicione o parêntese/colchete/chave de fechamento na próxima linha (**Figura 8**).

```
FILES = ('setup.cfg',)
```

Figura 7. Vírgulas finais são usadas ao criar uma tupla.

```
FILES = [  
    'setup.cfg',  
    'tox.ini',  
]  
initialize(FILES,  
    error=True,  
)
```

Figura 8. Múltiplas virgulas finais podem ser usadas em diferentes linhas.

6. COMENTÁRIOS

Os comentários em Python começam com o caractere cerquilha, #, e se estendem até o final visível da linha. Os comentários devem ser frases completas. Se um comentário for uma frase ou sentença, sua primeira palavra deverá ser maiúscula, a menos que seja um identificador que comece com uma letra minúscula.

Os comentários em bloco geralmente consistem em um ou mais parágrafos construídos a partir de frases completas, e cada frase deve terminar com um ponto final. Devem usar-se dois espaços após um ponto final de frase.

Bloquear comentários

Os comentários em bloco geralmente se aplicam a algum (ou a todos) o código que os segue e são recuados no mesmo nível desse código. Cada linha de um comentário em bloco começa com um # e um único espaço. Os parágrafos dentro de um comentário em bloco são separados por uma linha contendo um único #.

Comentários embutidos

Um comentário embutido é um comentário na mesma linha de uma declaração. Os comentários embutidos devem ser separados por pelo menos dois espaços da declaração. Eles devem começar com # e um espaço. Comentários embutidos podem ser desnecessários quando afirmarem o óbvio.

Strings de Documentação

Uma *docstring* Python é uma string usada para documentar um módulo, classe, função ou método Python, para que os programadores possam entender o que ele faz sem ter que ler os detalhes da implementação. Boas convenções de *docstring* são compiladas no PEP 257, no entanto, o mais importante é que o `"""` que termina uma *docstring* multilinha deve estar sozinho em uma linha (**Figura 9**). Para doutrinas de uma linha, manter o fechamento `"""` na mesma linha.

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

Figura 9. Fechamento de *docstrings* multilinha.

7. CONVENÇÕES DE NOMENCLATURA

Princípio primordial

Os nomes visíveis ao usuário como partes públicas da API devem seguir convenções que reflitam o uso, e não a implementação.

Descritivo: estilos de nomenclatura

Existem muitos estilos de nomenclatura diferentes. Ajuda ser capaz de reconhecer qual estilo de nomenclatura está sendo usado, independentemente da finalidade para a qual são usados. Os seguintes estilos de nomenclatura são comumente distinguidos:

- `b` (única letra minúscula)
- `B` (única letra maiúscula)
- `minúscula`
- `minúsculas_com_sublinhados`
- `MAIÚSCULAS`
- `UPPER_CASE_WITH_UNDERSCORES`
- `CapitalizedWords` (ou `CapWords`)
- `mixedCase` (difere de `CapitalizedWords` pelo caractere inicial minúsculo)
- `Capitalized_Words_With_Underscores`

Prescritivo: estilos de nomenclatura

- Nunca use os caracteres `'l'` (letra minúscula el), `'O'` (letra maiúscula oh) ou `'I'` (letra maiúscula olho) como nomes de variáveis de caractere único. Em algumas fontes, esses caracteres são indistinguíveis dos numerais um e zero. Quando tentado a usar `'l'`, use `'L'`.
- Os identificadores usados na biblioteca padrão devem ser compatíveis com ASCII.
- Os módulos devem ter nomes curtos e todos em letras minúsculas. Os sublinhados podem ser usados no nome do módulo se melhorarem a legibilidade. Os pacotes Python também devem ter nomes curtos, todos em letras minúsculas, embora o uso de sublinhados seja desencorajado.
- Os nomes das classes normalmente devem usar a convenção `CapWords`.
- Os nomes das funções devem estar em letras minúsculas, com palavras separadas por sublinhados conforme necessário para melhorar a legibilidade.

- Sempre usar *self* como primeiro argumento para métodos de instância.
- Sempre usar *cls* como o primeiro argumento dos métodos de classe.
- Nomes de métodos e variáveis de instância: devem seguir as mesmas convenções dos nomes de variáveis e funções, usando *Snake_case*. Por exemplo, `calcule_total()` e `instance_variable`.
- Constantes: As constantes devem ser escritas em *UPPER_CASE_WITH_UNDERSCORES*. Por exemplo, `MAX_CONNECTIONS` ou `DEFAULT_TIMEOUT`. Isso torna as constantes facilmente identificáveis, indicando valores que devem permanecer inalterados.
- 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 público; é mais fácil torná-lo público mais tarde do que tornar um atributo público não público. Atributos públicos são aqueles que você espera que clientes não relacionados de sua classe usem, com o seu compromisso de evitar alterações incompatíveis com versões anteriores. Atributos não públicos são aqueles que não se destinam ao uso por terceiros; você não oferece garantias de que os atributos não públicos não serão alterados ou mesmo removidos.
- Os atributos públicos não devem ter sublinhados iniciais.

8. RECOMENDAÇÕES DE PROGRAMAÇÃO

- Use funções e classes para encapsular código. Evite escrever grandes blocos de código fora de funções ou classes.
- Use os valores de parâmetro padrão com cuidado para evitar argumentos padrão mutáveis.
- Prefira exceções ao retorno de códigos de erro.
- Evite usar uma única instrução `return`, `break` ou `continue`. Use-os conforme necessário para melhorar a clareza.
- Evite expressões complexas; use variáveis intermediárias.
- Use `is` ou `not is` para comparações com `None`.

- Use a palavra-chave `in` para verificar se um valor está em uma sequência, em vez de usar um loop.
- Use os métodos `startswith()` e `endswith()` em vez de fatiar strings para verificações de prefixo/sufixo.
- Evite usar `==` para comparar com singletons como `None` ou `True`.

CONCLUSÃO

Ao longo do trabalho foram apresentadas recomendações e diretrizes obrigatórias para manter a clareza e o correto funcionamento de uma ferramenta poderosa como o Python.

Com isso obtemos orientação na hora de começar a escrever nossos códigos e entender melhor o trabalho de outros colaboradores. Principalmente, obtém-se maior clareza no que diz respeito ao estilo e estrutura do código, à elaboração de comentários, às convenções para escrever nomes corretamente, entre outras recomendações. O PEP-8 aborda desde aspectos básicos para iniciantes até alguns conceitos mais complexos que numa primeira abordagem biocientífica ao Python podem chegar a ser confusos, espera-se que no decorrer do aprendizado, com ajuda da prática, a compreensão do funcionamento deste universo de programação seja melhor. Este trabalho constituiu uma revisão para aprender sobre programação correta em Python e pode ser útil para qualquer programador que também esteja iniciando.

REFERÊNCIAS

VAN ROSSUM, Guido; WARREN, Barry. PEP 8 – Style Guide for Python Code. Python Software Foundation, 2001. Disponível em: <https://peps.python.org/pep-0008/>. Consultada em: 15 May 2024.

PYTHON SOFTWARE FOUNDATION. Python Tutorial. Disponível em: <https://docs.python.org/3/tutorial/>. Consultada em: 17 May 2024.