Python EFICIENTE

BOAS PRÁTICAS E PADRONIZAÇÃO DE CÓDIGO

POR MATHEUS BATTISTI
HORA DE CODAR

SOBRE O AUTOR

Matheus Battisti é desenvolvedor há mais de 6 anos e ama a tecnologia da informação como um todo.

Está sempre em busca de um aprendizado constante e evolutivo.

Atua como desenvolvedor Full Stack, desenvolvendo e-commerces.

É criador do Hora de Codar, que hoje passa de 20 mil alunos.

Python é uma de suas linguagens preferidas, já utilizou para Data Science como também desenvolvimento web.

Seu objetivo é sempre passar o máximo de conhecimento possível, unindo toda a experiência de seus estudos e também de sua carreira como desenvolvedor.

INTRODUÇÃO

Este eBook é destinado aos amantes de Python que querem melhorar o seu código.

Vou apresentar formas de escrever em Python que são consideradas as melhores.

Pois as mesmas são baseadas na PEP 8, uma convenção de estilo de código Python.

O que você aprender aqui poderá ser utilizado em qualquer software pois segue padrões dos mantenedores da linguagem.

Junto das explicações vou inserir exemplos práticos sempre que possível, para você assimilar melhor os conhecimentos.

Pronto para embarcar na aventura?

IDENTAÇÃO

Utilize sempre 4 espaços para identar as linhas de código, que fazem parte do próximo bloco.

```
nome = 'Matheus'

if nome == 'Matheus':
    print("Seu nome é Matheus")
    print("Idente com 4 espaços")
```

Linhas de continuação devem estar alinhadas verticalmente com o elemento que estão continuando.

```
def funcaoA(arg1, arg2,
            arg3, arg4,
            arg5):
    print(arg1)

funcaoA(1,2,3,4,5)
```

Perceba o alinhamento vertical dos argumentos que pertencem a mesma função.

Caso a identação de argumentos entre em conflito com a identação do bloco, você pode adicionar um comentário para não confundir os dois, veja:

IDENTAÇÃO

```
numberA = 1
numberB = 3

if (numberA and
    numberB):
    # Aqui devemos checar se os dois valores existem
    print(numberA + numberB)
```

Dessa forma é possível resolver o problema.

LARGURA MÁXIMA DE LINHA

As linhas nos programas de Python devem respeitar a largura máxima de 79 caracteres.

Este limite facilita a leitura humana do código, além de poder abrir várias janelas com diferentes arquivos e conseguir ler todos eles.

Também há alguns casos que terminais de SSH limitam a tela para 80 caracteres, então 79 encaixa neste número e nada será cortado.

QUEBRA DE LINHA ANTES DE OPERADOR BINÁRIO

Sempre que houve um operador binário, por exemplo uma soma, e precisarmos quebrar linha, esta quebra deve ser feita antes do operador, veja:

```
numA = 2
numB = 5
numC = 6

soma = (numA
        + numB
        * numC)
```

LINHAS EM BRANCO

Você deve separar as classes e as funções de nível superior com duas linhas.

Já os demais códigos devem ser separados apenas por uma quebra de linha.

IMPORTANDO MÓDULOS

A importação de módulos deve ser feita de modo que cada um fique em sua própria linha de código.

E também os imports devem ficar sempre no topo do código, veja um exemplo:

```
import os
import sys
import builtins
```

STRINGS: ASPAS SIMPLES OU DUPLAS

Em Python usar aspas simples ou duplas não faz diferença alguma no código.

Porém escolha uma das formas e mantenha este padrão em todo o seu software.

ESPAÇOS EM BRANCO

É recomendado evitar espaços em brancos em algumas situações.

1. Depois de um parênteses de abertura;
2. Antes de um parênteses de fechamento;
3. Depois de uma vírgula ou ponto e vírgula;
4. Antes dos argumentos de uma função;
5. Antes de abrir colchetes;
6. Mais de um espaço na declaração de variáveis;

Veja estes casos de **forma errada** abaixo:

```
# 1
foo( x,y)

# 2
foo(x,y )

# 3
if x == 4 : print x , y ; x , y = y , x

# 4
foo (x,y)

# 5
conj ['teste'] = 2

# 6
teste = 2
x     = 2
```

COMENTÁRIOS

Lembre-se de sempre atualizar seus comentários quando alterar o código.

Cuidado com os comentários inline, eles podem atrapalhar em vez de ajudar, pois distraem o programador.

Escreva comentários preferencialmente em inglês pois é uma linguagem universal.

CONVENÇÃO DE NOMES

Nunca utilize l (L minúsculo), O maiúsculo, ou i maiúsculo.

Os módulos que você cria devem ter apenas uma palavra no nome, e todas as letras em minúsculo.

As classes devem utilizar capitalização em todos as palavras, ex: MinhaClasse

As funções devem ter sempre o nome em letras minúsculas e as palavras separadas por underline, ex: minha_funcao_teste

CONVENÇÃO DE NOMES

As constantes devem ser sempre em letras maiúsculas e as palavras separadas por underline, ex: NUM_MAXIMO

RECOMENDAÇÕES PARA PROGRAMAR

> Opte pelo operador **is not** em vez de **is ... not**

> Seja consistente no retorno das funções, ou retorne expressões em todos os returns, ou em nenhuma delas

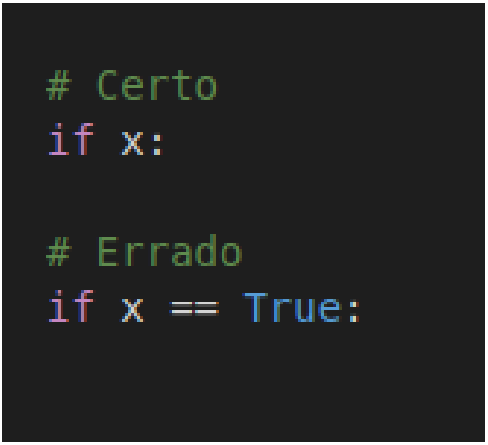
> Use **''.startswith**() e **''.endswith**() em vez de outras operações para checar sufixos e prefixos

> Comparações de objetos sempre devem utilizar a função **isistance**()

> Não compare booleanos com True ou False

RECOMENDAÇÕES PARA PROGRAMAR

> Não compare booleanos com True ou False



> As exceções de código devem preferencialmente derivar de **Exception** não de **BaseException**

> Quando utilizar **try catch**, tente manter o bloco de try sempre enxuto, apenas com o necessário

> Use os **métodos de string** em vez do módulo **string**

DICAS GERAIS

Modo Zen do Python: para saber como fazer um código Pythônico, utilize o módulo this, deste jeito:

E você terá este retorno:

The Zen of Python, by Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Estas indicações foram feitas por Tim Peters, com o intuito de expressar como código em Python deve ser escrito

DICAS GERAIS

No Python podemos criar uma instrução que é chamada de atribuição encadeada

Onde podemos atribuir valores para várias variáveis de uma vez só, veja:

```
a = b = c = 10

print(c)
```

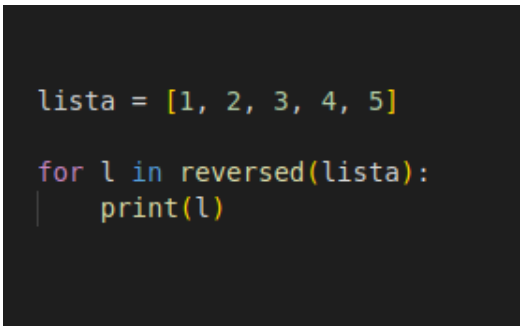
Outro conceito interessante do Python é o **Swap Variables**

Onde podemos trocar o valor de duas variáveis de uma forma muito fácil, veja:

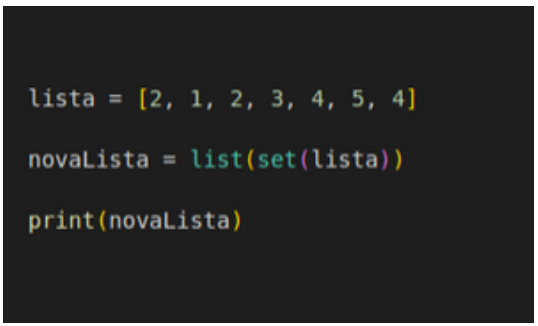
```
x = 5
y = 10

x, y = y, x

print(x) # 10
```

DICAS GERAIS

No Python também é possível fazer uma iteração reversa por uma lista, veja:



Podemos também remover duplicados facilmente de uma lista, veja:



Obrigado

Enquanto houver pessoas dispostas
a aprender, eu estarei criando
conteúdo para enriquecer ainda
mais os seus conhecimentos

Matheus Battisti

Hora de Codar