



# Boas práticas: Formatação, Linting e Documentação

## EditorConfig

[Padrões para "matching"](#)

## Estilo de código Python (Style Guide)

[PEP8 – Guia de Estilo para Código Python](#)

## Black

[Utilizando black via linha de comando](#)

[Configurando black no VSCode](#)

[pyproject.toml](#)

[Outras ferramentas de formatação](#)

[Settings do VSCode para organizar imports automaticamente \(instalar isort\)](#)

## Análise estática de código

[Linters \(\[pylint\]\(#\) e \[flake8\]\(#\)\)](#)

[Type-checking](#)

## Documentação

### Documentação de Código

[One-line docstrings](#)

[Multi-line docstrings](#)

[Integração com editor](#)

### Documentação API

[Especificação OpenAPI](#)

[Ferramentas para escrever OpenAPI specs e gerar documentação](#)

[Exemplo de documentação do projeto tamarcado](#)

## GitHub Pages

## Git Commit Hooks

## Conclusão

## Exercícios



Palavras-chave: EditorConfig, Black, Flake8, TypeHint, Docstrings, OpenAPI, GitHub Pages, Git Commit Hooks.

# EditorConfig

<https://editorconfig.org/#overview>

Como o nome já diz: **arquivo de configuração para editores de texto**. *Language-agnostic*, ou seja, qualquer projeto/linguagem pode utilizar o EditorConfig para garantir um padrão consistente entre diferentes colaboradores utilizando *diferentes editores de texto*.

## ▼ Arquivo de exemplo ( [.editorconfig](#) )

Link para o Gist: <https://gist.github.com/gcrsaldanha/01a9da7a22f77d0ea106d9a013533609>

```
# Baseado no exemplo do site EditorConfig: https://EditorConfig.org
# Alguns editores têm integração nativa com EditorConfig. VSCode não é um deles.
# Plugin VSCode: https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig

# Arquivo EditorConfig raiz
# Podemos ter mais de um por projeto (para pastas específicas),
# mas não é muito comum.
root = true

[*] # Match em todos os arquivos
end_of_line = lf # Quebra de linha de sistema Unix (LF - Line Feed - \n)
# Outras opções: crlf (Mac), cr (Windows \r)
insert_final_newline = true # Boa prática: toda linha deve terminar com uma quebra de linha

[*.js,py,md] # Match em qualquer arquivo com essas extensões
charset = utf-8 # charset padrão

[*.py] # Match apenas em arquivos .py.
# PRIORIDADE é top-down! Não pela regra mais específica.
# Ou seja, a regra definida por último tem prioridade.
indent_style = space
indent_size = 4

[*.js]
indent_style = space
indent_size = 2 # Para JavaScript vamos definir apenas 2 espaços de indentação.

[*.md]
indent_style = space
indent_size = 2

# Tab indentation (no size specified)
[Makefile]
indent_style = tab # Usar o caractere tabulação (\t) para indentação em arquivos Makefile.
# Mesmo que a gente aperte TAB em um arquivo .py, na verdade, vai inserir ESPAÇOS.
# Não precisamos especificar o tamanho da indentação. Ou seja, será mantido como está ou como definido pelo editor.

# Podemos sobreescriver arquivos em uma pasta específica
[lib/**.js] # /**.js => recursivo, ou seja, qualquer arquivo *.js em qualquer subdiretório de lib/
indent_style = space
indent_size = 2

# Arquivos específicos:
[django.yml, pytest.ini]
indent_style = space
indent_size = 2
```

## Padrões para "matching"

### WILDCARD PATTERNS

Special characters recognized in section names for wildcard matching:

*	Matches any string of characters, except path separators (/)
**	Matches any string of characters
?	Matches any single character
[ name ]	Matches any single character in <i>name</i>
[ !name ]	Matches any single character not in <i>name</i>
{s1,s2,s3}	Matches any of the strings given (separated by commas) <b>(Available since EditorConfig Core 0.11.0)</b>
{num1..num2}	Matches any integer numbers between <i>num1</i> and <i>num2</i> , where <i>num1</i> and <i>num2</i> can be either positive or negative

Special characters can be escaped with a backslash so they won't be interpreted as wildcard patterns.

<https://editorconfig.org/#file-format-details>

## Estilo de código Python (*Style Guide*)

### PEP8 – Guia de Estilo para Código Python

Link: <https://peps.python.org/pep-0008/>

Estabelece regras sobre estilo de código Python. Isso inclui tanto questões sobre utilizar "espaço" ou "tabulação" para indentação, quanto onde `imports` devem aparecer no código-fonte.

Também diz o que é opcional (aspas duplas ou aspas simples).



Isso não significa que seu projeto seja **obrigado a seguir 100% das regras**, é apenas um *guia* e deve ser encarado como tal. Entretanto, você deve ter boas razões para *fugir* ao guia.

A vantagem de seguir o guia é que facilita muito a contribuição com outras pessoas, já que o *padrão* é o código Python estar de acordo com a PEP8.

## **Black**

Link: <https://black.readthedocs.io/en/stable/index.html>

“Any color you like.”

É um *code formatter*, ou formatador de código. Segue a PEP8 e é uma solução *opinionated*, ou seja, toma diversas decisões por padrão para precisar do mínimo de configuração necessária.

O propósito do Black é evitar que desenvolvedores fiquem discutindo sobre um estilo específico, na dúvida, utilize o estilo padrão do próprio Black.

### **Utilizando black via linha de comando**

```
pip install black  
black --check <path>.py  
black <path>.py
```

No próprio site tem uma seção para integração com editores:

<https://black.readthedocs.io/en/stable/integrations/editors.html>

Instruções para VSCode: [https://code.visualstudio.com/docs/python/editing#\\_formatting](https://code.visualstudio.com/docs/python/editing#_formatting)

### **Configurando black no VSCode**

formatting

Usuário Workspace

Extensões (23)

- C/C++ (1)
- C# configuration (2)
- GitLens — Git su... (12)
- Current Line Bla... (1)
- Status Bar Blame (2)
- Hovers (1)
- Views (6)
- Gutter Blame (1)
- Date & Times (1)

Python (7)

- Vim (1)

Python > Formatting: Autopep8 Args

Arguments passed in. Each argument is a separate item in the array.

Adicionar Item

Python > Formatting: Autopep8 Path (🔗 Sincronização: Ignorada)

Path to autopep8, you can use a custom version of autopep8 by modifying this setting to include the full path

autopep8

Python > Formatting: Black Args

Arguments passed in. Each argument is a separate item in the array.

Adicionar Item

Python > Formatting: Black Path (🔗 Sincronização: Ignorada)

Path to Black, you can use a custom version of Black by modifying this setting to include the full path.

black

Python > Formatting: Provider

Provider for formatting. Possible options include 'autopep8', 'black', and 'yapf'.

black

Python > Formatting: Yapf Args

Arguments passed in. Each argument is a separate item in the array.

Adicionar Item

Configurar o black como “provedor de formatação”

format on save

Usuário Workspace

Editor de Texto (2)

Formatação (2)

Editor: Format On Save

Formatar um arquivo ao salvar. Um formatador precisa estar disponível, o arquivo não deve ser salvo após o atraso e o editor não deve estar desligando.

Editor: Format On Save Mode

Controla se o formato no salvamento formata o arquivo inteiro ou somente as modificações. Aplica-se somente quando Editor: Format On Save é habilitado.

file

Configurar auto-formatação ao salvar o arquivo

## pypackage.toml

- Arquivo de configuração para diversas ferramentas Python

- Estabelecido na [PEP 518](#).
- Inclusive, pode substituir o `pytest.ini`. Mais informações [aqui](#).

```
[tool.black]
line-length = 120
target-version = ['py310']
--exclude = '**/migrations/*.py
```

## Outras ferramentas de formatação

`autoflake8`: remover imports e variáveis não utilizados (não funciona tão bem no VSCode).

```
autoflake --in-place --remove-all-unused-imports --remove-unused-variables <path>.py
```

`isort`: ordenar imports (padrão, libs, local)

```
isort autoflake --in-place --remove-all-unused-imports --remove-unused-variables <path>.py
```



Se estiver usando VSCode: Paleta de Comandos > Organizar Imports

## Settings do VSCode para organizar imports automaticamente (instalar isort)

[Paleta de Comandos > Configurações \(JSON\)](#)

```
"editor.codeActionsOnSave": {
    "source.organizeImports": true
}
```

## Análise estática de código

### Linters (`pylint` e `flake8`)

- Encontrar possíveis problemas através de uma análise estática do seu código.
- Métodos muito complexos, variáveis não utilizadas, etc.
  - Complexidade Ciclomática é uma aula por si só! Recomendo pesquisar sobre 😊
- Configuração extensiva. Mas com o tempo auxilia bastante na qualidade do código.

```
pip install pylint
pip install flake8
pylint <path>.py
flake8 <path>.py
```

- Configurar no VSCode: [Paleta de Comandos > Selecionar Linter > pylint/flake8](#)



Eu particularmente prefiro o flake8 porque o pylint acaba sendo **muito verboso** e difícil de configurar.

#### ▼ Arquivo de configuração para flake8

```
[flake8]
extend-ignore = E501 # Ignora max-line
max-line-length=120
exclude=
    .git,
    __pycache__,
    */migrations/**,
    tamarcado/settings*,
max-complexity = 10
```

## Type-checking

- Verificador de tipagem
- Ajuda muito a evitar bugs (`NullPointer`)
- Usar tipagem facilita muito a vida do desenvolvedor - ordem de argumentos numa função, por exemplo.

```
pip install flake8-annotations
flake8 <path>.py
```



Outra opção é utilizar o `mypy`, mas ele requer mais configuração.

## Documentação

### Documentação de Código

Utilize as `docstrings` (PEP257: <https://peps.python.org/pep-0257/>):

#### One-line docstrings

```
def is_primo(n):
    """Verifica se o número é primo utilizando o método da raiz quadrada."""
    ...
```

#### Multi-line docstrings

```
class IsOwnerOrCreateOnly(permissions.BasePermission):
    """
        Permite escrita para qualquer usuário, leitura apenas para owner.

        Permissão que verifica se usuário é o "dono" (prestador de serviço) pelo username passado
        ou se é um método para criar agendamento (POST).
    
```

```
"""
...
def get_horarios_disponiveis(data: date) -> Iterable[datetime]:
    """
    Verifica os horários disponíveis para uma determinada data.

    Caso a data passada seja um feriado, nenhum horário disponível é retornado. Para essa consulta
    é utilizada a API de feriados...
    """
```

 Algumas ferramentas como [Sphinx](#) utilizam uma sintaxe mais verbosa ([reStructuredText](#)) para gerar documentação a partir dos comentários. Hoje em dia, com *type hint*, isso se tornou menos necessário. Ver a [documentação do Flask](#).

## ▼ Exemplo simples RST docstrings

```
def my_function(my_arg, my_other_arg):
    """
    A function just for me.

    :param my_arg: The first of my arguments.
    :param my_other_arg: The second of my arguments.

    :returns: A message (just for me, of course).
    """
    pass
```

## Integração com editor

```
(function) get_horarios_disponiveis: (data: date) -> Iterable[datetime]
Verifica os horários disponíveis para uma determinada data.

Caso a data passada seja um feriado, nenhum horário disponível é retornado. Para essa consulta é utilizada a API de feriados...
st[get_horarios_disponiveis(data)]) You, há 4 semanas • Implementa get_horarios API ...
```

## Documentação API

### Especificação OpenAPI

- Definir as URLs, versão da API, query params, request, response, etc.
- [A BrasilAPI utiliza essa especificação](#)

 OpenAPI é a especificação. Swagger oferece um conjunto de ferramentas para a geração do documento de especificação e de um site em HTML. [Redocly](#) é outra opção (é a que a BrasilAPI usa).

## Ferramentas para escrever OpenAPI specs e gerar documentação

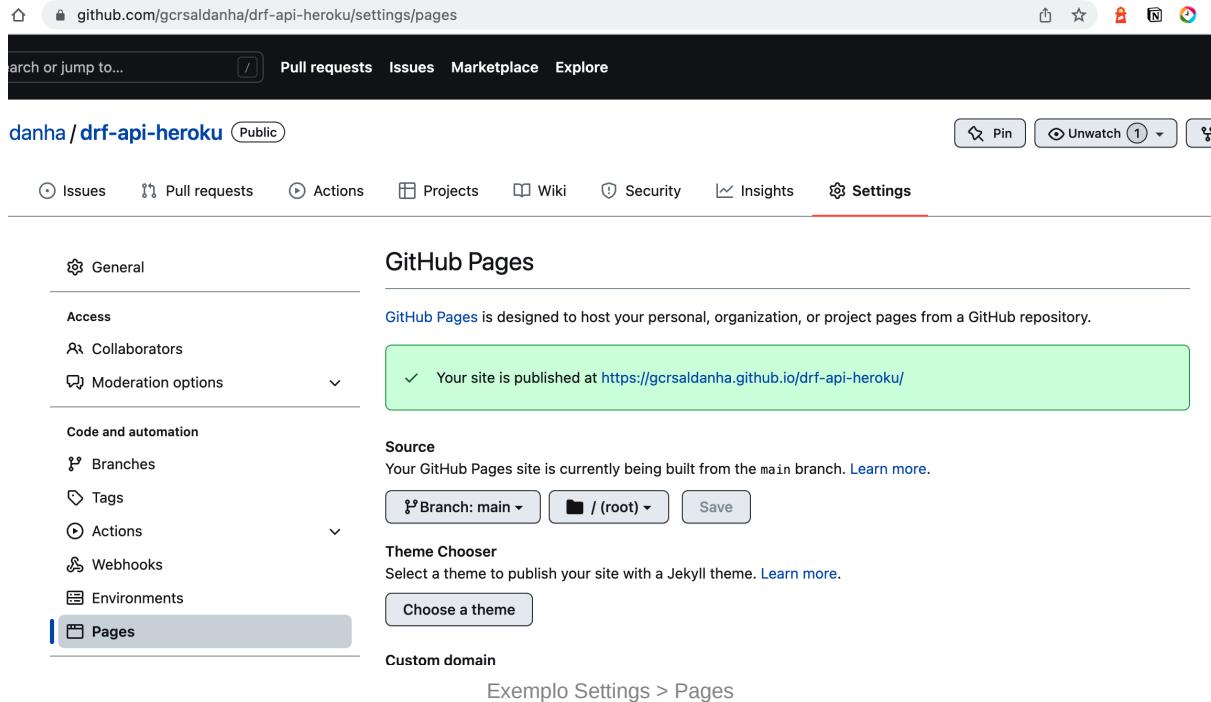
- [Swagger](#)
- [Redoc online renderer](#)
- [Redoc CLI](#) (pode passar o link de um gist!)
- [VSCode Plugin](#)

## Exemplo de documentação do projeto tamarcado

[openapi.yml](#)

## GitHub Pages

- Criar um arquivo `index.html` ou `index.md` ou `README.md` na raiz do repositório.
- Na página do GitHub do seu repositório, acessar [Settings > Pages](#)
  - Selecionar a branch (main) e salvar



The screenshot shows the GitHub repository settings page for 'gcrsaldanha/drf-api-heroku'. The 'Pages' tab is highlighted with a red underline. On the left sidebar, the 'Pages' tab is also highlighted with a blue bar. The main content area displays the GitHub Pages configuration, showing a green box indicating the site is published at <https://gcrsaldanha.github.io/drft-api-heroku/>.

- Acessar o link indicado!



Você pode observar o status da *build* na aba de Actions!

- Você também pode selecionar diferentes temas!
  - Selecionar um tema vai automaticamente criar um *commit* de configuração usando Jekyll.
- ▼ Adicionar configuração ao seu HTML

```
---  
layout: default  
---  
<h1>Pagina inicial!</h1>
```

## Git Commit Hooks

The screenshot shows the VSCode settings interface with the 'exclude' section selected. On the left, there's a sidebar with categories like 'Comumente Usado', 'Editor de Texto', 'Recursos', 'Extensões', etc. The main panel shows the 'Local History: Exclude' configuration, which allows users to exclude files or folders from the local history. A 'Files: Exclude' section is also visible, showing the pattern '\*\*/.git'. At the bottom, there's a note about the .git folder being excluded by default.

Por padrão o VSCode exclui/esconde a pasta .git, remover dessa lista para conseguir visualizar

- Caso esteja utilizando VSCode, remover a pasta `.git` das opções de arquivos excluídos (File Exclude)
- ▼ Adicionar o arquivo `pre-commit` na pasta hooks

```
#!/bin/sh  
  
# Se estiver usando windows  
# #!C:/Program\ Files/Git/usr/bin/sh.exe (caminho para o sh.exe do Git Bash)  
# Créditos: https://www.tygertec.com/git-hooks-practical-uses-windows/  
  
# Outra opção para windows, criar um arquivo chamado pre-commit.ps1 com o script  
# E aqui adicionar a linha:  
# exec powershell.exe -NoProfile -ExecutionPolicy Bypass -File ".\git\hooks\pre-commit.ps1"  
  
black --check .
```

- Ao tentar fazer um commit, o script `pre-commit` é executado e só permite o commit caso nenhum erro ocorra.



Caso você queira ignorar os pre-commit checks, basta fazer o commit com a opção `--no-verify` ou `-n`:  
`git commit -n -m "Ignorando pre-commits"`

## Conclusão

- Aprender a configurar essas ferramentas leva tempo, e normalmente cada projeto já tem uma configuração padrão e você não precisa se preocupar.
- O mais importante é entender os conceitos e quando você for enviar algum projeto de exercício, **sempre inclua formatters e linters**.

- O módulo 17 vai falar sobre CI/CD, e vamos ver como executar algumas verificações (exemplo: rodar testes) *no próprio repositório* – porque atualmente o usuário pode simplesmente fazer um commit/push com `--no-verify`.

## Exercícios

- Executar os testes do projeto quando o usuário tentar fazer um `push`.
- Criar documentação para a API de listar prestadores
  - `GET /api/prestadores`

▼ Resposta

```
[
  {
    "id": 1,
    "username": "gcrsaldanha",
    "agendamentos": [
      {
        "id": 1,
        "prestador": "gcrsaldanha",
        "cancelado": false,
        "data_horario": "2024-01-10T10:00:00Z",
        "nome_cliente": "Carl",
        "email_cliente": "bob@codar.me",
        "telefone_cliente": "+5521999978888"
      },
      {
        "id": 2,
        "prestador": "gcrsaldanha",
        "cancelado": false,
        "data_horario": "2024-02-10T10:00:00Z",
        "nome_cliente": "Carl",
        "email_cliente": "bob@codar.me",
        "telefone_cliente": "+5521999978888"
      }
    ]
  }
]
```



Utilize `$ref` para referenciar os `Agendamentos` dentro de cada prestador.

- **Desafio:** instale a `Redoc CLI` e gere um arquivo HTML com a documentação a partir da especificação `openapi.yml`. Suba essa documentação como uma página no GitHub Pages.