

# Desenvolvimento Aberto



**Documentação de API e linters**

Igor dos Santos Montagner ( [igorsm1@insper.edu.br](mailto:igorsm1@insper.edu.br) )

# Código vs software profissional

Os seguintes pontos transformam um código que fiz para mim em algo útil para outras pessoas

1. Traduções e internacionalização (datas)
2. Documentação de usuário
3. Documentação de desenvolvimento

# Código vs software profissional

Os seguintes pontos transformam um código que fiz para mim em algo útil para outras pessoas

1. Traduções e internacionalização (datas)
2. Documentação de usuário
3. Documentação de desenvolvimento
  - Estrutura de projeto
  - **API**

# Hoje

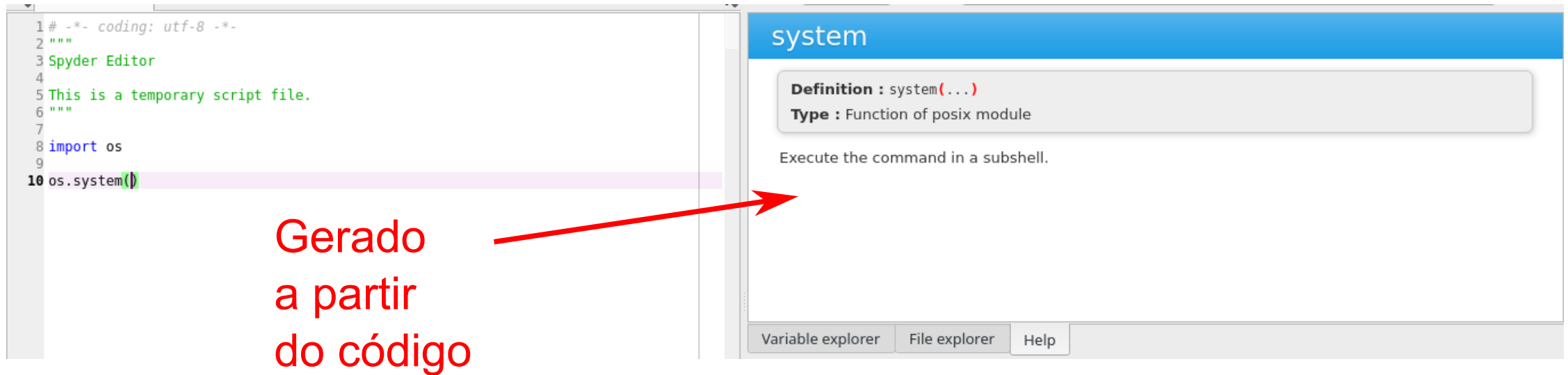
- Documentação de API usando
  - pydoc
  - sphinx-autodoc
- Padrões de formatação de código
  - linters
  - PEP8

# Documentação de API

**Objetivo:** explicar o funcionamento das funções, classes e módulos de um programa.

- Focado em detalhes
- Documenta os argumentos esperados e em quais situações a função funciona
- Tipicamente obtida direto do código

# Documentação de API



The image shows a screenshot of the Spyder IDE interface. On the left, a code editor displays a Python script with the following content:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 import os
9
10 os.system()
```

The line `os.system()` is highlighted in purple. A red arrow points from the text "Gerado a partir do código" (Generated from the code) to the `os.system()` line. On the right, the API documentation for the `system` function is displayed. The title "system" is in a blue header. Below it, the definition is shown as `system(...)`, and the type is listed as "Function of posix module". The description states: "Execute the command in a subshell." At the bottom of the right panel, there are tabs for "Variable explorer", "File explorer", and "Help".

Gerado a partir do código

**system**



**Definition :** `system(...)`

**Type :** Function of posix module

Execute the command in a subshell.

Variable explorer File explorer Help

# Documentação de API

 SciPy.org 

SciPy.org Docs NumPy v1.15 Manual NumPy Reference Routines Linear algebra ( `numpy.linalg` )

index next previous

## numpy.dot

`numpy.dot(a, b, out=None)`

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using `matmul` or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to `multiply` and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where  $M \geq 2$ ), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:]* b[k,:,m])
```

**Parameters:**

- a : array\_like**  
First argument.
- b : array\_like**  
Second argument.
- out : ndarray, optional**  
Output argument. This must have the exact kind that would be returned if it was not used. In particular, it must have the right type, must be C-contiguous, and its dtype must be the dtype that

Previous topic  
[Linear algebra \( `numpy.linalg` \)](#)

Next topic  
[numpy.linalg.multi\\_dot](#)

Quick search

ref

# Ferramentas

- Python:
  - pydoc, **sphinx-apidoc**
- C/C++
  - Doxygen
- Java
  - Javadoc



# Padrões de codificação

```
def funcaoQueFazAlgo(a, b):  
    print('Algo!!')  
  
def outra_funcao(arg1, arg2):  
    print("Outra funcao!", arg1+arg2)  
  
funcaoQueFazAlgo(1, 2)  
outra_funcao(3, 4)
```

# Padrões de codificação

```
[igor@haute-normandie 09-api-padroes-de-codigo]$ pylint porco.py
No config file found, using default configuration
***** Module porco
W:  4, 0: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
C:  7, 0: Trailing whitespace (trailing-whitespace)
C:  1, 0: Missing module docstring (missing-docstring)
C:  3, 0: Function name "funcaoQueFazAlgo" doesn't conform to snake_case naming style (invalid-name)
C:  3, 0: Argument name "a" doesn't conform to snake_case naming style (invalid-name)
C:  3, 0: Argument name "b" doesn't conform to snake_case naming style (invalid-name)
C:  3, 0: Missing function docstring (missing-docstring)
W:  3,21: Unused argument 'a' (unused-argument)
W:  3,24: Unused argument 'b' (unused-argument)
C:  6, 0: Missing function docstring (missing-docstring)

-----
Your code has been rated at -6.67/10 (previous run: -5.00/10, -1.67)
```

# Padrões de codificação

- Cada projeto tem o seu
- Algumas linguagens tem um estilo padrão
  - Python - PEP8
- Ferramentas ajudam a conferir (forçar) um estilo específico

# Ferramentas

- Python: pylint
- C/C++: splint, cppchecker, gcc (opções -Wall, -Wextra)
- Java: flag `-Xlint`
- Javascript: ESLint, TSLint (typescript)

Ajudam a manter código limpo e legível. Podem ser plugadas no seu editor/IDE favorito.

# Testes automatizados

**Ideia:** escrever um programa que verifica se um outro programa responde como esperado

- Definir situações a serem testadas ...
- e o resultado esperado em cada situação

# Testes automatizados

## Não ajudam:

- a revelar novos bugs
- a garantir que um software é livre de bugs

## Ajudam

- a evitar que bugs descobertos voltem
- a evitar que mudanças não intencionais quebrem código que estava funcionando.
- a documentar em quais situações o software funciona.

# Testes unitários

**Ideia:** dada uma função, verificar se ela devolve o valor esperado para um certo conjunto de parâmetros.

- Testa as funções de maneira **isolada**
- **Cobertura:** porcentagem das linhas de código que é executada durante os testes de unidade.
- Serve como documentação da função

# Testes unitários - pytest

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

That's it. You can now execute the test function:

```
$ pytest
===== test session starts =====
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F [100%]

===== FAILURES =====
_____ test_answer _____

    def test_answer():
>         assert func(3) == 5
E         assert 4 == 5
E         + where 4 = func(3)

test_sample.py:5: AssertionError
===== 1 failed in 0.12 seconds =====
```



**O quê eu preciso testar?**

Ninguém sabe de verdade....

# Atividade prática

Transformar um código perdido em um projeto "completo"

**Skill:** Projeto profissional

**Proof:** url da página do projeto criada por vocês.

Mais instruções no roteiro da aula.

# Desenvolvimento Aberto



**Documentação de API + testes**

Igor dos Santos Montagner ( [igorsm1@insper.edu.br](mailto:igorsm1@insper.edu.br) )