

# Decoradores

**Guilherme Arthur de Carvalho**

Analista de sistemas

**<https://linktr.ee/decarvalhogui>**

# Objetivo Geral

Conhecer os decoradores e como utilizá-los.

# Pré-requisitos

- Conhecimento básico em Python.

# Percurso

## **Etapa 1**

## **Decoradores em Python**

## Etapa 1


# Decoradores em Python

# Recapitulando

Funções em Python são objetos de primeira classe. Isso significa que **as funções podem ser passadas e usadas como argumentos.**

tmp >  decoradores.py > ...

```
1 def dizer_oi(nome):  
2     return f"Oi {nome}"  
3  
4  
5 def incentivar_aprender(nome):  
6     return f"Oi {nome}, vamos aprender Python juntos!"  
7  
8  
9 def mensagem_para_guilherme(funcao_mensagem):  
10     return funcao_mensagem("Guilherme")  
11  
12  
13 mensagem_para_guilherme(dizer_oi)  
14 mensagem_para_guilherme(incentivar_aprender)  
15
```

●  \$ python decoradores.py  
Oi Guilherme  
Oi Guilherme, vamos aprender Python juntos!

# Inner functions

É possível definir funções dentro de outras funções . Tais funções são chamadas de funções internas .



```
1  def pai():
2      print("Escrevendo da pai() função")
3
4      def filho1():
5          print("Escrevendo da filho1() função")
6
7      def filho2():
8          print("Escrevendo da filho2() função")
9
10     filho2()
11     filho1()
12
13
14 pai()
```

```
● $ python decoradores.py
Escrevendo da pai() função
Escrevendo da filho2() função
Escrevendo da filho1() função
```

# Retornando funções de funções

Python também permite que você use funções como valores de retorno.

```
1  def calcular(operacao):
2      def somar(a, b):
3          return a + b
4
5      def subtrair(a, b):
6          return a - b
7
8      if operacao == "+":
9          return somar
10     else:
11         return subtrair
12
13
14 resultado = calcular("+")(1, 3)
15 print(resultado)
16
```

# Decorador simples

Agora que entendemos que funções são como qualquer outro objeto em Python, podemos seguir em frente e ver a mágica que é o decorador Python.

```
1  def meu_decorador(funcao):
2      def envelope():
3          print("Faz algo antes de executar a função.")
4          funcao()
5          print("Faz algo depois de executar a função.")
6
7      return envelope
8
9
10 def ola_mundo():
11     print("Olá mundo!")
12
13
14 ola_mundo = meu_decorador(ola_mundo)
15 ola_mundo()
16
```

# Açúcar Sintático!

O Python permite que você **use decoradores de maneira mais simples com o símbolo @.**

```
1  def meu_decorador(funcao):
2      def envelope():
3          print("Faz algo antes de executar a função.")
4          funcao()
5          print("Faz algo depois de executar a função.")
6
7      return envelope
8
9
10 @meu_decorador
11 def ola_mundo():
12     print("Olá mundo!")
13
14
15 ola_mundo = meu_decorador(ola_mundo)
16 ola_mundo()
```

# Funções de decoração com argumentos

Podemos usar **\*args** e **\*\*kwargs** na função interna, com isso ela aceitará um número arbitrário de argumentos posicionais e de palavras-chave.



```
1 def duplicar(func):
2     def envelope(*args, **kwargs):
3         func(*args, **kwargs)
4         func(*args, **kwargs)
5
6     return envelope
7
8
9 @duplicar
10 def aprender(tecnologia):
11     print(f"Estou aprendendo {tecnologia}")
12
13
14 aprender("Python")
15
```

# Retornando valores de funções decoradas

O decorador pode decidir se retorna o valor da função decorada ou não. Para que o valor seja retornado a função de **envelope** deve retornar o valor da função decorada.

```
1 def duplicar(func):
2     def envelope(*args, **kwargs):
3         func(*args, **kwargs)
4         return func(*args, **kwargs)
5
6     return envelope
7
8
9 @duplicar
10 def aprender(tecnologia):
11     print(f"Estou aprendendo {tecnologia}")
12     return tecnologia.upper()
13
14
15 tecnologia = aprender("Python")
16 print(tecnologia)
17
```

# Introspecção

Introspecção é a capacidade de um objeto saber sobre seus próprios atributos em tempo de execução.

```
>>> print
<built-in function print>
>>> print.__name__
'print'
>>> aprender
<function duplicar.<locals>.envelope at 0x7fa70ba4fec0>
>>> aprender.__name__
'envelope'
>>>
```

```
1 import functools
2
3
4 def duplicar(func):
5     @functools.wraps(func)
6     def envelope(*args, **kwargs):
7         func(*args, **kwargs)
8         return func(*args, **kwargs)
9
10    return envelope
11
12
13 @duplicar
14 def aprender(tecnologia):
15     print(f"Estou aprendendo {tecnologia}")
16     return tecnologia.upper()
17
18
19 tecnologia = aprender("Python")
20 print(tecnologia)
>>> aprender
<function aprender at 0x7f4fe6db7ec0>
>>> aprender.__name__
'aprender'
>>>
```

# Percurso

~~Etapa 1~~

~~Decoradores em Python~~

# Links Úteis

- <https://github.com/digitalinnovationone/trilha-python-dio>



# Dúvidas?

- > Fórum/Artigos
- > Comunidade Online (Discord)

