Live de Python

Decoradores

Obrigado!

apoia.se/livedepython

```
z4r4tu5tr4 at babbage in ~/git/apoiadores
$ python apoiadores.py
Amaziles Carvalho    Andre Machado        Bruno Guizi          Carlos Augusto
Cleber Santos        David Reis           Dayham Soares        Diego Ubirajara
Edimar Fardim        Eliabe Silva         Elias Soares         Emerson Lara
Fabiano Silos        Fabiano Teichmann    Fabiano Gomes        Fernando Furtado
Fábio Serrão         Gleison Oliveira     Humberto Rocha       Jean Vetorello
Johnny Tardin        Jonatas Oliveira     Jonatas Simões       João Lugão
Jucélio Silva        Júlia Kastrup        Leon Teixeira        Magno Malkut
Maria Boladona       Matheus Francisco    Nilo Pereira         Pablo Henrique
Paulo Tadei          Pedro Alves          Rafael Galleani      Regis Santos
Renan Moura          Renato Santos        Rennan Almeida       Rodrigo Vaccari
Sérgio Passos        Thiago Araujo        Tiago Cordeiro       Vergil Valverde
Vicente Marcal       Wander Silva         Welington Carlos     Wellington Camargo
Welton Souza         William Oliveira     Willian Gl           Yros Aguiar
```

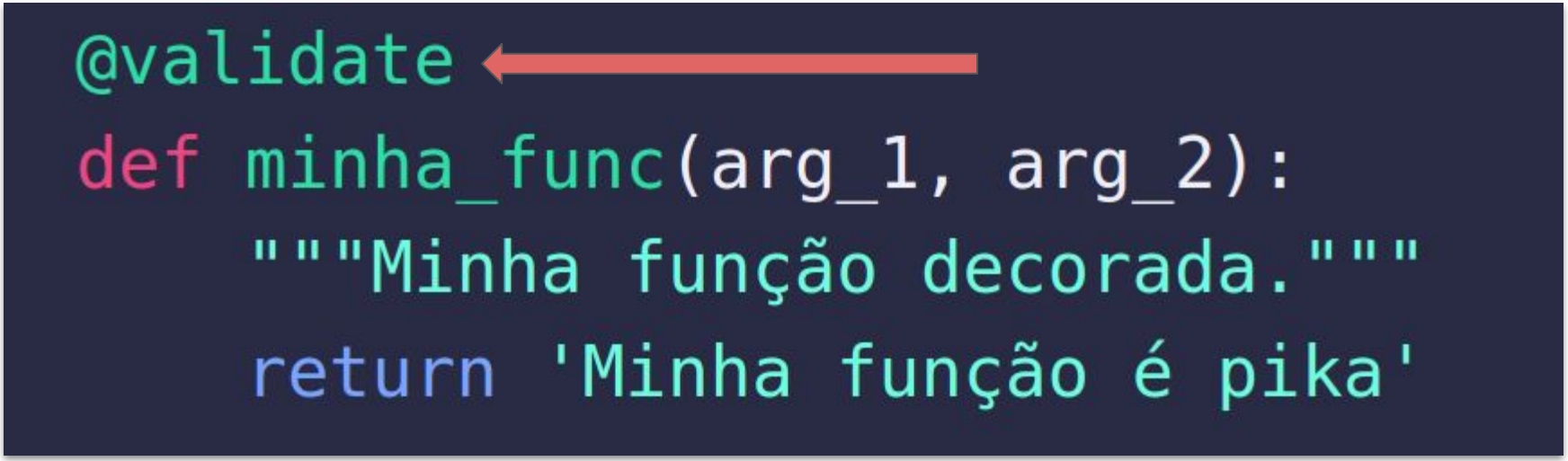
Roteiro

- Qual a cara desse decorador aí?
- Uma firula teórica
  - Nested Functions
  - Closures
- Fazendo o nosso próprio decorador
- Decifrando linha a linha
- Decoradores úteis e prontos (se sobrar tempo)

Qual a cara desse decorador? [slide_1.py]

```
@validate
def minha_func(arg_1, arg_2):
    """Minha função decorada."""
    return 'Minha função é pika'
```

Qual a cara desse decorador? [slide_1.py]



Qual a cara desse decorador? [slide_1.py]

```
@validate
def minha_func(arg_1, arg_2):
    """Minha função decorada."""
    return 'Minha função é pika'
```

Qual a cara desse decorador? [slide_1.py]

```
 7    @validate
 8    def minha_func(arg_1, arg_2):
 9        """Minha função decorada."""
10        return 'Minha função é pika'
11
12
13    print(minha_func())
14
```

```
z4r4tu5tr4 at babbage in ~/live87
$ python slide_1.py
42
```

Uma firula teórica

Nested functions [slide_2.py]

Sim, funções dentro de funções

```
def func_externa(val):
    """Função com outra definida internamente."""
    def procedimento_oculto():
        return val

    print('Antes do oculto')
    print(val)
    print('Depois do oculto')
```

Nested functions [slide_2.py]

Sim, funções dentro de funções

```
13
14      func_externa(10)
15
```

```
z4r4tu5tr4 at babbage in ~/live87
$ python slide_2.py
Antes do oculto
10
Depois do oculto
```

Live de Python

Closures [slide_3.py]

Sim, aqui as coisas começam a ficar um pouco mais cabeludas. Mas sério, é fácil de entender. Você vai sacar só de olhar.

Closures são funções com **escopo estendido**. Elas conseguem ter a “visão além do alcance”.

Closures [slide_3.py]

```
def media_acumulada():
    valores = []

    def calcula_media(valor):
        valores.append(valor)
        return sum(valores)/len(valores)

    return calcula_media
```

Closures [slide_3.py]

```
11    media = media_acumulada()
12    media(2)
13    media(2)
14    media(2)
15    media(2)
16    media(2)
17    print(media(500))
18
```

```
z4r4tu5tr4 at babbage in ~/live87
$ python slide_3.py
85.0
```

```
    2
    2
    2
    2
    2
+ 500
---------
  510
```

510 / 6 = 85

Live de Python

Closures para nerds

Closures [slide_3.py] - Nerds Only

```
def media_acumulada():
    valores = []

    def calcula_media(valor):
        valores.append(valor)
        return sum(valores)/len(valores)

    return calcula_media
```

Closure

Variável livre

Live de Python

Closures [slide_3.py] - Nerds Only

Tá, mas onde isso fica armazenado?

```
In [1]: media_acumulada.__code__.co_varnames
Out[1]: ('calcula_media',)

In [2]: media.__code__.co_varnames
Out[2]: ('valor',)

In [3]: media.__closure__
Out[3]: (<cell at 0x7ff012a50d08: list object at 0x7ff012a2f348>,)
```

Closures [slide_4.py] - Nerds Only

Apesar do escopo ser compartilhado e as variáveis livres ainda estarem no contexto, objetos imutáveis não podem ser modificados pela função interna

```
def contador_pika():
    cont = 0

    def contador():
        cont += 1
        return cont

    return contador
```

Live de Python

```
In [1]: contador = contador_pika()

In [2]: contador()
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-2-32794a151af3> in <module>
----> 1 contador()

~/live87/slide_4.py in contador()
      3
      4     def contador():
----> 5         cont += 1
      6         return cont
      7

UnboundLocalError: local variable 'cont' referenced before assignment
```

Closures [slide_4.py] - Nerds Only

Agora (faz tempo), no python 3 existe a palavra nonlocal faz esse tipo de modificação.

```
def contador_pika():
    cont = 0

    def contador():
        cont += 1
        return cont

    return contador
```

```
def contador_pika():
    cont = 0

    def contador():
        nonlocal cont
        cont += 1
        return cont

    return contador
```

Live de Python

Closures [slide_4.py] - Nerds Only

Agora (faz tempo), no python 3 existe a palavra nonlocal faz esse tipo de modificação.

```
def contador_pika():
    cont = 0

    def contador():
        cont += 1
        return cont

    return contador
```

```
def contador_pika():
    cont = 0

    def contador():
        nonlocal cont
        cont += 1
        return cont

    return contador
```

Live de Python


QUANDO VOCÊ DIZ PYTHON 2
imgflip.com

```
In [1]: contador = contador_pika()

In [2]: contador()
Out[2]: 1

In [3]: contador()
Out[3]: 2
```

Rodando com nonlocal

Falou bonito, mas não falou porra nenhuma ainda

Um decorador passo a passo

Um decorador é uma função* que recebe uma função e pode manipular suas entradas e saídas.

....

Vamos fazer de boas

Problema 1

Crie uma função que some **somente** inteiros