Profiling

Live de Python #204

Roteiro

1. Profiling? É de comer?

Uma visão geral sobre o que é e onde se esconde

2. Timers

Começando do começo

3. Memória

Investigando consumo de memória

4. cProfile

Indo bem fundo

5. Vizualiação

Facilitando nosso entendimento sobre o código com gráficos

6. Bonus

Aquelas libs que salvam

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Acássio Anjos, Ademar Peixoto, Alexandre Harano, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Ana Padovan, Andre Azevedo, André Rocha, Aquiles Coutinho, Arnaldo Turque, Bloquearsites Farewall, Bruno Barcellos, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, César Almeida, Christiano Morais, Clara Battesini, Daniel Freitas, Daniel Haas, Danilo Segura, Dartz Dartz, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Douglas Bastos, Douglas Braga, Douglas Martins, Douglas Zickuhr, Eli Júnior, Emerson Rafael, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Evandro Avellar, Everton Silva, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Rozas, Fernando Sousa, Flávio Meira, Flavkaze Flavkaze, Franklin Silva, Gabriel Barbosa, Gabriel Simonetto, Geandreson Costa, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Gustavo Dettenborn, Gustavo Suto, Heitor Fernandes, Henrique Junqueira, Igor Taconi, Ismael Ventura, Israel Gomes, Italo Silva, Jair Andrade, Jairo Lenfers, Janael Pinheiro, Jean Marcio, João Lugão, Johnny Tardin, Jonatas Leon, Jonatas Oliveira, Jônatas Silva, Jorge Plautz, Jose Mazolini, Juan Gutierrez, Juliana Machado, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Botassio, Leandro Miranda, Leonardo Cruz, Leonardo Mello, Leonardo Nazareth, Lucas Adorno, Lucas Mello, Lucas Mendes, Lucas Oliveira, Lucas Polo, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Luiz Paula, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mario Deus, Mateus Lisboa, Matheus Silva, Matheus Vian, Mírian Batista, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nicolas Teodosio, Osvaldo Neto, Otávio Barradas, Patricia Minamizawa, Patrick Felipe, Paulo Braga, Paulo Tadei, Pedro Duarte, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Rafael Lopes, Rafael Rodrigues, Rafael Romão, Ramayana Menezes, Reinaldo Silva, Renan Moura, Renato Veirich, Riverfount Riverfount, Robson Maciel, Rodrigo Brandao, Rodrigo Ferreira, Rodrigo Freire, Rodrigo Junior, Rodrigo Vaccari, Rodrigo Vieira, Rogério Sousa, Ronaldo Silva, Rui Jr, Samanta Cicilia, Sara Selis, Thalysson Bogéa, Thiago Araujo, Thiago Borges, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Thiago Oliveira, Thiago S, Thiago Souza, Tiago Minuzzi, Tony Dias, Victor Wildner, Vinícius Bastos, Vinicius Figueiredo, Vítor Gomes, Vitor Luz, Vlademir Souza, Vladimir Lemos, Walter Reis, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Yury Barros

Obrigado você

Profiling

O que é?

Segundo a documentação

The Python Profilers

Código-fonte: Lib/profile.py and Lib/pstats.py

Introduction to the profilers

`cProfile` and `profile` provide *deterministic profiling* of Python programs. A *profile* is a set of statistics that describes how often and for how long various parts of the program executed. These statistics can be formatted into reports via the `pstats` module.

Um *profile* é um conjunto de estatisticas que descrevem com que frequência e por quanto tempo varias partes do programa foram executadas

Profiling

É a arte de encontrar "gargalos" nas aplicações.

- Saber quanto tempo demora
- Quanto de memória está sendo consumido
- Que parte do sistema demora mais que outras
- etc...

Profiling

É a arte de encontrar "gargalos" nas aplicações.

- Saber quanto tempo demora
- Quanto de memória está sendo consumido
- Que parte do sistema demora mais que outras
- etc...

Nosso objetivo hoje é dar uma abordagem genérica, que você consiga aplicar em problemas diferentes. Web, DS, desktop, ...

Profiling

Isso vai aparecer com vários nomes por aí, traduzidos ou não, apelidos

- profiling
- "profilar"
- criar perfil
- Perfilar
- ...

Um exemplo polêmico [exemplo_01.py]

```python
def append_em_lista():
    lista = []

    for el in range(10_000):
        conta = (el ** 3) / 7
        lista.append(conta)

    return lista
```

VS

```python
def list_comp():
    return [
      (el ** 3) / 7
      for el in range(10_000)
    ]
```

Um exemplo mais polêmico ainda [exemplo_02.py]

```python
l = []

if l == []:
    # Seu código aqui!
    ...
```

VS

```python
l = []

if l:
    # Seu código aqui!
    ...
```

Desempenho não quer dizer NADA!

Nota IMPORTANTE!

Pense nisso depois de

- O código funcionar
- O código estar testado
- Já estiver validado

Otimização prematura é a raiz de todo mal

Donald Knut

Outra nota mais importante

Tipos de profiling

- Amostras
- Memória
- Determinísticos (trace) - Assunto pra outra live

A forma mais básica
de se testar algo

timers

import timeit

O python tem uma biblioteca nativa para fazer medições do **tempo** em que alguma coisa demora para ser executada.

- **timeit**: repede por diversas vezes um determinado bloco de código e diz o tempo final que levou
- **repeat**: repede por diversas vezes um determinado bloco de código e retorna todos os tempos para podermos analizar

timeit

```python
from timeit import timeit

def uma_função_qualquer():
    ...

tempo_total_em_segundos = timeit(
  'uma_função_qualquer()',
  globals=globals(),
  number=100
)
```

iPython / Jupyter

```
In [2]: def foo():
   ...:     ...

In [3]: %timeit foo()
48.9 ns ± 2.58 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

repeat

```python
from timeit import repeat

def uma_função_qualquer():
    ...

lista_com_todos_os_tempos = repeat(
    'uma_função_qualquer()',
    globals=globals(),
    repeat=100
)
```

Dica de ouro, ou não

```python
from statistics import mean
from timeit import repeat

def means(stmt, r=100):
  result = repeat(stmt, globals=globals(), repeat=r)
  return f'- Min: {min(result)}, Max: {max(result)}, Mean: {mean(result)}'

means('foo()')
'''
Min: 0.046434504009084776,
Max: 0.07729666600062046,
Mean: 0.047875771270337285
'''
```

No shell

```
1   python -m timeit -s 'import exemplo_00' 'exemplo_00.foo()'
2   5000000 loops, best of 5: 63.9 nsec per loop
```

Memória

Entendendo o consumo de memória

Consumo de memória

Uma das coisas importantes, pela dinamicidade de todo o código é entender se existem gargalos de memória.

Para isso temos uma biblioteca externa chamada **memory profiler**

Ela analisa um bloco de código, linha a linha com um decorador e nos diz o consumo de memória

pip install memory_profiler

instalando

Um exemplo de alto consumo de memória [exemplo_03.py]

```python
def carregar_arquivo():
    # Arquivo de 2.9 Mb
    with open('br-utf8.txt') as file:
        conteudo = file.read()

    return conteudo

carregar_arquivo()
```

Adicionando o profiler

```
1  from memory_profiler import profile
2
3  @profile
4  def carregar_arquivo():
5      # Arquivo de 2.9 Mb
6      with open('br-utf8.txt') as file:
7          conteudo = file.read()
8
9      return conteudo
10
11 carregar_arquivo()
```

O resultado

```
○ → python exemplo_03.py
Filename: /home/dunossauro/git/live-beta/live_203/exemplo_03.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     4     18.2 MiB     18.2 MiB           1   @profile
     5                                         def carregar_arquivo():
     6     21.1 MiB      0.0 MiB           2       with open('br-utf8.txt') as file:
     7     21.1 MiB      3.0 MiB           1           conteudo = file.read()
     8
     9     21.1 MiB      0.0 MiB           1       return conteudo
```

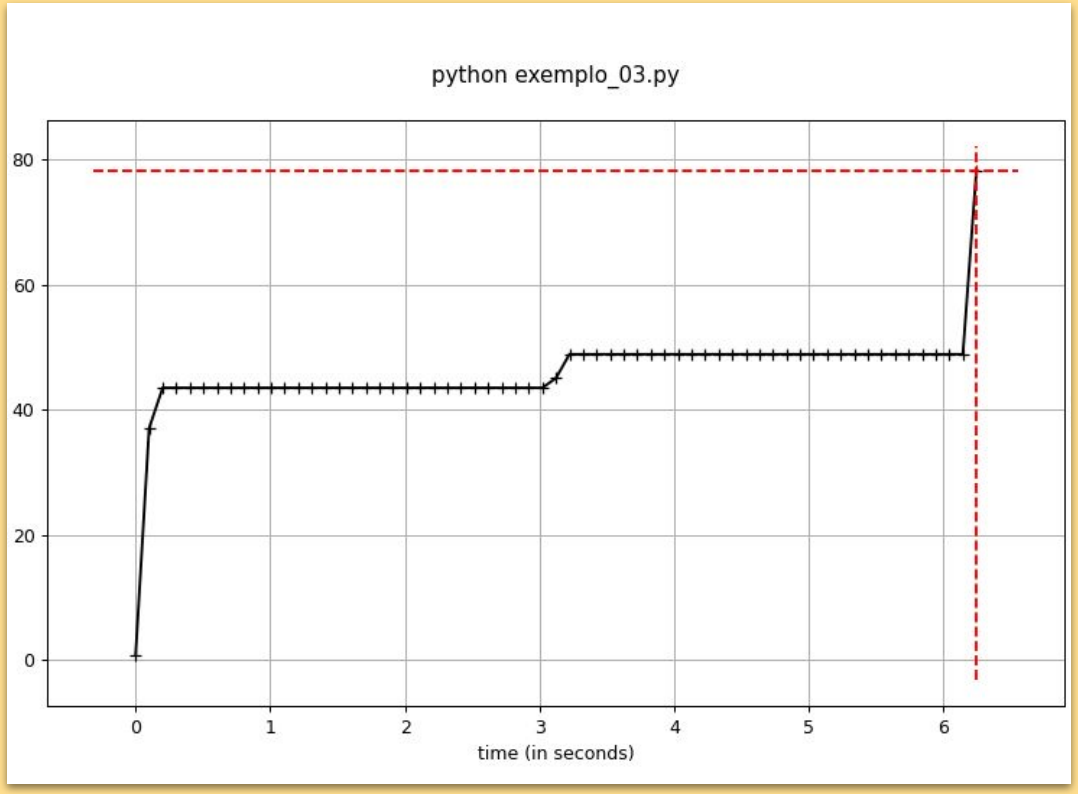
Total de memória

Incrementado na linha

pip install matplotlib

instalando

Vendo os resultados




```
mprof clean # limpa execuções passadas
mprof run --python python exemplo_03.py
mprof plot
```

Um caso real de consumo exagerado de memória

Mostrar o ldp_photo - Live 201/202

Descendo um pouco o nível

C Profile

cProfile

Quando falamos em profilers em python, talvez o cProfile seja a primeira coisa que vem a mente, ele é nativo e cumpre o papel de maneira muito honesta.

```
1   from cProfile import run
2
3   run('1 + 1')
```

O resultado

```
○ → python exemplo_04.py
         3 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

O resultado


Tempo total gasto
Onde está sendo executado
python exemplo_04
3 function c
Ordered by: standard name
ncalls tottime percall cumtime percall filename:lineno(function)
1 0.000 0.000 0.000 0.000 <string>:1(<module>)
1 0.000 0.000 0.000 0.000 {built-in method builtins.exec}
1 0.000 0.000 0.000 0.000 {method 'disable' of '_lsprof.Profiler' objects}
Quantas vezes foi chamado
tottime / ncalls

pstats [exemplo_04.py]

```
1   import cProfile
2   import pstats
3
4   prof = cProfile.Profile()
5
6   prof.enable()
7   # Seu código
8   prof.disable()
9
10  stats = pstats.Stats(prof).sort_stats('ncalls')
11  stats.print_stats()
```

O poder está no shell

```
1   pythom -m cProfile -o output.stats exemplo_04.py
2   python -m pstats output.stats
```

Vamos ver no ldp_photo

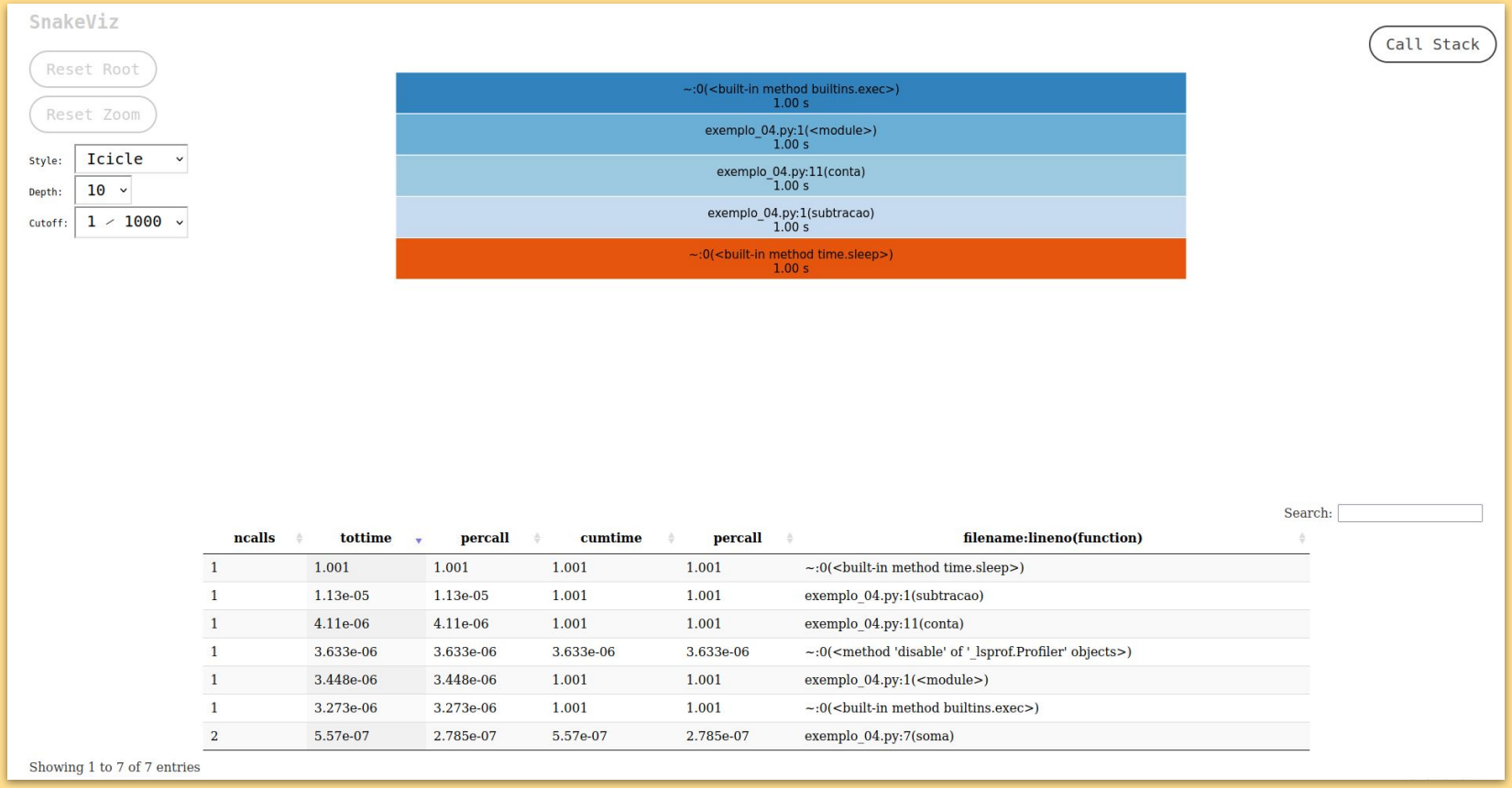
Um problema complexo

snakeviz

Pra facilitar a visualização do pstats, que fica inviável em aplicações complexas temos o snakeviz

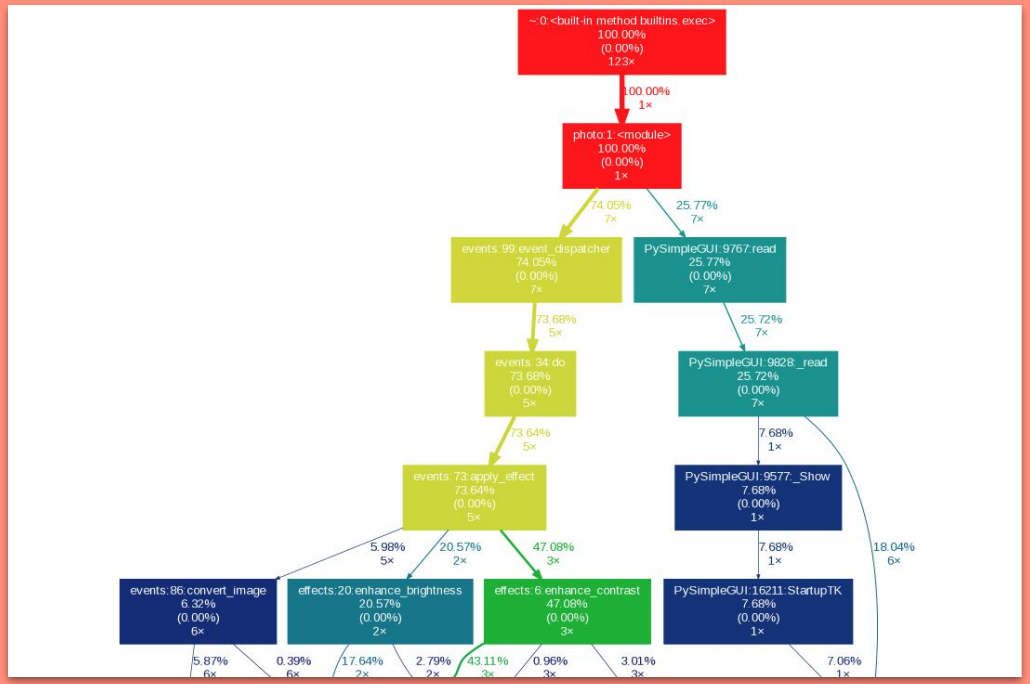
```
1   pip install snakeviz
2   snakeviz output.stats
```

snakeviz


SnakeViz
Reset Root
Reset Zoom
Style: Icicle
Depth: 10
Cutoff: 1 / 1000
Call Stack
~:0(<built-in method builtins.exec>)
1.00 s
exemplo_04.py:1(<module>)
1.00 s
exemplo_04.py:11(conta)
1.00 s
exemplo_04.py:1(subtracao)
1.00 s
~:0(<built-in method time.sleep>)
1.00 s
Search:
ncalls
tottime
percall
cumtime
percall
filename:lineno(function)
1 1.001 1.001 1.001 1.001 ~:0(<built-in method time.sleep>)
1 1.13e-05 1.13e-05 1.001 1.001 exemplo_04.py:1(subtracao)
1 4.11e-06 4.11e-06 1.001 1.001 exemplo_04.py:11(conta)
1 3.633e-06 3.633e-06 3.633e-06 3.633e-06 ~:0(<method 'disable' of '_lsprof.Profiler' objects>)
1 3.448e-06 3.448e-06 1.001 1.001 exemplo_04.py:1(<module>)
1 3.273e-06 3.273e-06 1.001 1.001 ~:0(<built-in method builtins.exec>)
2 5.57e-07 2.785e-07 5.57e-07 2.785e-07 exemplo_04.py:7(soma)
Showing 1 to 7 of 7 entries

Uma árvore da execução

Se precisar de ver o fluxo de execuções e o quanto cada coisa está consumindo. Pode usar o **gprof2dot**



gprof2dot

```
pip install gprof2dot
gprof2dot -f pstats output.pstats | dot -Tpng -o out.png
eog out.png  # eog é meu visualiador de imagens
```

Perfomance em tempo real

```
Collecting samples from 'python photo.py' (python v3.10.4)
Total Samples 9000
GIL: 1.00%, Active: 100.00%, Threads: 1

  %Own   %Total  OwnTime  TotalTime  Function (filename)
  0.00% 100.00%   0.000s    54.55s   <module> (photo.py)
  0.00% 100.00%   0.000s    53.70s   event_dispatcher (app/events.py)
  0.00% 100.00%   0.000s    51.87s   do (app/events.py)
  0.00% 100.00%   0.000s    51.83s   apply_effect (app/events.py)
  0
  0
 62
  0
  0
  0
 37
  0
  0
  0
  0
  0.00%   0.00%   0.000s     2.19s   enhance (PIL/ImageEnhance.py)
  0.00%   0.00%   0.000s     1.76s   __init__ (PIL/ImageEnhance.py)
  0.00%   0.00%   0.190s     1.64s   convert (PIL/Image.py)
  0.00%   0.00%   0.000s     1.58s   undo (app/events.py)
  0.00%   0.00%   0.000s    0.720s   read (PySimpleGUI/PySimpleGUI.py)
```

```
pip install py-spy
py-spy top -- python photo.py
```

As dicas do Joe Rickerby xD

Outros

Só o necessário - pyinstrument

```
Program: photo.py

33.934 <module>  <string>:1
   [4 frames hidden]  <string>, runpy
      33.934 _run_code  runpy.py:63
      └─ 33.934 <module>  photo.py:1
         ├─ 26.488 event_dispatcher  app/events.py:99
         │  └─ 26.274 do  app/events.py:34
         │     └─ 26.261 apply_effect  app/events.py:73
```

```
pip install pyinstrument
pyinstrument photo.py
```

```
         │           └─ 1.967 thumbnail  PIL/Image.py:2409
         │                 [11 frames hidden]  PIL, <built-in>
         └─ 7.369 read  PySimpleGUI/PySimpleGUI.py:9767
               [135 frames hidden]  PySimpleGUI, tkinter, <built-in>, ins...
```

Eliot

Um logger que te conta o que está acontecendo

```
pip install eliot eliot-tree
python exemplo_05.py
eliot-tree eliot.log
```

```
1  from eliot import log_call, to_file
2
3  to_file(open('eliot.log', 'w'))
4
5  @log_call
6  def minha_função():
7      ...
```

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Referências

- https://pythonspeed.com/pygotham19/
- https://docs.python.org/3/