

## # Manual de Utilização: `py-spy` e `cProfile`

Este guia detalha a utilização das ferramentas de profiling em Python, `py-spy` e `cProfile`, que permitem identificar gargalos de desempenho em aplicações Python.

### Uso do `py-spy`

#### 1. Instalação

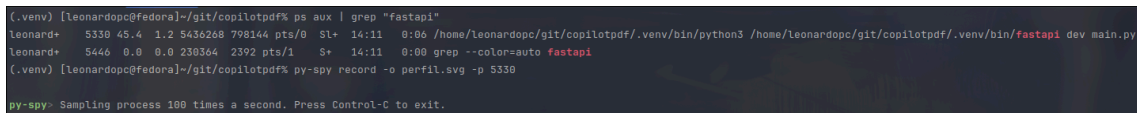
Instale o `py-spy` usando `pip`:

```
```bash
pip install py-spy
```
```

#### 2. Gravação de desempenho

Execute o `py-spy` para gerar um perfil visual do desempenho:

```
```bash
py-spy record -o segmentation.svg -- python3 main.py
```
```



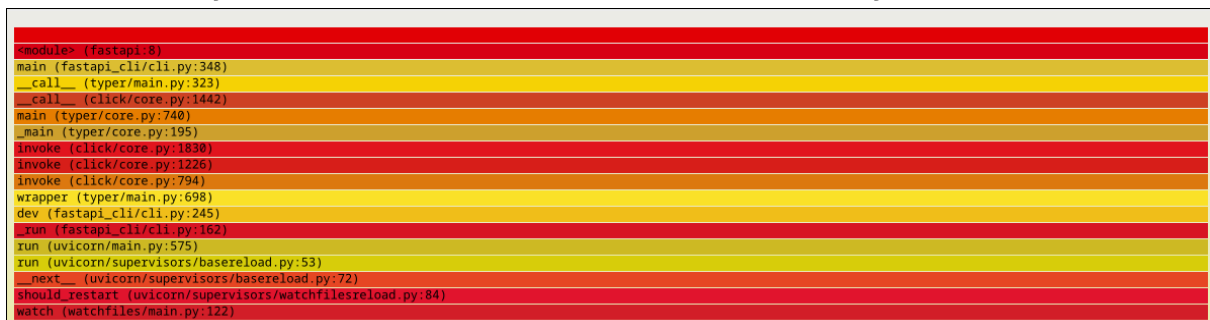
```
(.venv) [leonardopc@fedora]~/git/copilotpdf% ps aux | grep "fastapi"
leonard+  5330 45.4  1.2 5436268 798144 pts/0  S+   14:11   0:06 /home/leonardopc/git/copilotpdf/.venv/bin/python3 /home/leonardopc/git/copilotpdf/.venv/bin/fastapi dev main.py
leonard+  5446  0.0  0.0 230364 2392 pts/1    S+   14:11   0:00 grep --color=auto fastapi
(.venv) [leonardopc@fedora]~/git/copilotpdf% py-spy record -o perfil.svg -p 5330

py-spy> Sampling process 100 times a second. Press Control-C to exit.
```

\* O arquivo `segmentation.svg` será gerado com o perfil coletado.

#### 3. Visualização dos resultados

- \* Abra o arquivo `segmentation.svg` em seu navegador ou gerenciador de arquivos.
- \* Clique nas funções exibidas para explorar seus tempos de execução.



Neste caso aqui podemos ver que tanto o módulo base do fastapi, invoke do core da aplicação, restart e watch estão sendo os maiores utilizadores de recursos da aplicação, sendo Should\_restart a aplicação de reboot do fastapi e o watch a que identifica as mudanças nos arquivos da pasta, logo, em um servidor correto é retirar-las

#### 4. Perfil avançado (chamadas nativas)

Para capturar detalhes mais profundos, incluindo código nativo, utilize:

```
```bash
py-spy record -o segmentation.svg --native -- python3 main.py
```
```



## Uso do `cProfile`

### Exemplo prático

A seguir, usaremos como exemplo a função `processor.segmented\_pdf`. Em outros casos, substitua pelo nome adequado da sua função.

#### 1. Importação

```
```python
import cProfile
```
```

#### 2. Execução e geração do perfil

Execute o profiling diretamente no código:

```
```python
cProfile.run("processor.segmented_pdf(file_path)")
```
```

#### 3. Salvando resultados

Salve os resultados em um arquivo `.prof` para análise posterior:

```
```python
cProfile.run("processor.segmented_pdf(file_path)", "segmentation.prof")
```
```

#### 4. Análise dos resultados

Abra o arquivo `segmentation.prof` em seu editor ou utilize o módulo `pstats`:

```
```python
import pstats
ps = pstats.Stats("segmentation.prof")
```
```

#### ### 5. Visualização detalhada com `pstats`

Para visualizar estatísticas específicas ordenadas por tempo cumulativo (`cumtime`):

```
```python
ps.strip_dirs().sort_stats("cumtime").print_stats("processor.segmented_pdf")
```
```