LU07.A11 - Den echten Hotspot finden

Tippen Sie zuerst, welche Funktion die Zeit frisst. Messen Sie danach mit `cProfile` - und vergleichen Sie mit Ihrer Vermutung.

Ausgangslage

```
import random
random.seed(1)

gesperrt = [f"user{i}" for i in range(5000)]
anfragen = [f"  User{random.randrange(10000)}  " for _ in range(3000)]


def normalisieren(name):
    return name.strip().lower()


def ist_gesperrt(name):
    return name in gesperrt


def formatieren(treffer):
    return "\n".join(f"gesperrt: {t}" for t in treffer)


def pruefen(namen):
    return [n for n in namen if ist_gesperrt(normalisieren(n))]
```

Detaillierte Aufgabenstellung

1. **Tippen Sie zuerst.** Notieren Sie schriftlich, welche der drei Funktionen Ihrer Meinung nach am meisten Zeit braucht - und warum. Erst danach messen.
2. Profilen Sie den Aufruf:

```
import cProfile, pstats
cProfile.run("pruefen(anfragen)", "profil.stats")
pstats.Stats("profil.stats").sort_stats("cumulative").print_stats(8)
```

1. Lesen Sie `ncalls`, `tottime` und `cumtime` und bestimmen Sie den Hotspot.
2. Ändern Sie **nur** den Hotspot. Messen Sie mit `timeit` vorher und nachher und prüfen Sie das Ergebnis mit `assert`.
3. Optimieren Sie anschliessend zusätzlich die Funktion, die Sie ursprünglich im Verdacht hatten. Messen Sie, was das bringt.

Zu beantwortende Fragen

- Warum wirkt `normalisieren` teuer, obwohl es das nicht ist?
- Was sagt `ncalls` über die Struktur des Programms aus?
- Was ist der Unterschied zwischen `tottime` und `cumtime` bei `pruefen`?
- Wie viel Prozent der Gesamtzeit haben Sie mit der zweiten Optimierung noch gewonnen - und war der Aufwand gerechtfertigt?

M323-LU07, M323-D1A

© Kevin Maurizi

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m323/learningunits/lu07/aufgaben/profiling?rev=1788945535

Last update: **2026/09/09 11:18**