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.

Repository: m323-lu07-a11-profiling

Ausgangslage

```
GESPERRT = [f"user{i}" for i in range(5000)]


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


def ist_gesperrt(name):
    return name in GESPERRT


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

Die Anfragen kommen in unsauberer Schreibweise herein (`„ User1234 “`), deshalb wird jede zuerst normalisiert.

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. Dieser Schritt ist der Kern der Aufgabe.
2. **Profilen:** `python main.py` führt die vorbereitete Funktion `profil()` aus.
3. Lesen Sie `ncalls`, `tottime` und `cumtime` und bestimmen Sie den Hotspot. Den Hotspot suchen Sie in `tottime`, nicht in `cumtime` - sonst steht immer die äusserste Funktion zuoberst.
4. Ändern Sie **nur** den Hotspot. Messen Sie mit `timeit` vorher und nachher, prüfen Sie das Ergebnis mit `assert`.
5. Profilen Sie erneut. Was steht jetzt zuoberst, und wie gross ist die Gesamtlaufzeit noch?

Regeln

`normalisieren` und `ist_gesperrt` bleiben als **benannte Funktionen** erhalten; zwei Tests prüfen das. Die naheliegende zweite Optimierung - alle Hilfsfunktionen in die Comprehension ziehen - bringt weniger als ein Prozent und kostet zwei einzeln testbare Funktionen. Das ist ein Refactoring in die falsche

Richtung, verkauft als Optimierung.

Zu beantwortende Fragen (BEFUND.md)

- 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 brächte die zweite Optimierung noch - und wäre der Aufwand gerechtfertigt?

Bewertung

Teil	Punkte
Tests (`main_test.py`)	9
pylint (`main.py`)	5

M323-LU07, M323-D1A