LU07f - Leistung messen

Ab hier geht es nicht mehr um Lesbarkeit, sondern um Laufzeit - das zweite Standbein von Kompetenzband D. Die Regel dazu ist kurz: **Erst messen, dann ändern, dann nachmessen.**

Ohne Messung ist es keine Optimierung, sondern eine Vermutung. Und Vermutungen über Laufzeit sind erstaunlich oft falsch - auch bei erfahrenen Entwicklerinnen und Entwicklern. Für **D1A** verlangt das Portfolio ausdrücklich eine Messung vorher und nachher.

timeit: einzelne Varianten vergleichen

`timeit` führt ein Stück Code mehrfach aus und misst die Zeit. Mehrfach deshalb, weil ein einzelner Durchlauf zu stark schwankt.

```
import timeit, random
random.seed(42)

gesperrt_liste = [f"user{i}" for i in range(10000)]
gesperrt_set = set(gesperrt_liste)
anfragen = [f"user{random.randrange(20000)}" for _ in range(2000)]

def mit_liste():
    return sum(1 for a in anfragen if a in gesperrt_liste)

def mit_set():
    return sum(1 for a in anfragen if a in gesperrt_set)

assert mit_liste() == mit_set()          # gleiches Ergebnis, sonst ist der
Vergleich wertlos

print(min(timeit.repeat(mit_liste, number=1, repeat=5)))
print(min(timeit.repeat(mit_set,   number=1, repeat=5)))
```

```
0.1070      # Liste
0.000068    # Set        -> rund 1500-mal schneller
```

Drei Dinge machen die Messung erst brauchbar:

- **`repeat` und `min`**: Der schnellste Durchlauf ist der aussagekräftigste, weil er am wenigsten durch andere Prozesse gestört wurde.
- **Gleiche Daten** für beide Varianten, und zwar realistische - eine Liste mit 10 Einträgen misst nichts.
- **`assert` auf gleiches Ergebnis.** Eine schnellere Funktion, die etwas anderes liefert, ist keine Optimierung.

Auf der Kommandozeile geht es auch ohne Skript:

```
python -m timeit -s "d = list(range(10000))" "9999 in d"
python -m timeit -s "d = set(range(10000))" "9999 in d"
```

cProfile: den Hotspot finden

`timeit` vergleicht zwei Varianten, die Sie schon im Verdacht haben. `cProfile` sagt Ihnen, *wo* die Zeit überhaupt hingeht.

```
import cProfile, pstats

cProfile.run("pruefen(anfragen)", "profil.stats")

p = pstats.Stats("profil.stats")
p.sort_stats("cumulative").print_stats(6)
```

```
         12004 function calls in 0.104 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.002    0.002    0.104    0.104 prof_demo.py:12(pruefen)
     3000    0.099    0.000    0.099    0.000 prof_demo.py:9(ist_gesperrt)
     3000    0.002    0.000    0.003    0.000 prof_demo.py:6(normalisieren)
     3000    0.001    0.000    0.001    0.000 {method 'strip' of 'str'
objects}
```

So lesen Sie die Spalten:

Spalte	Bedeutung
`ncalls`	wie oft die Funktion aufgerufen wurde
`tottime`	Zeit **in** der Funktion selbst, ohne Unterfunktionen
`cumtime`	Zeit inklusive allem, was sie aufruft

Der Befund ist eindeutig: `ist_gesperrt` verbraucht 0.099 der 0.104 Sekunden. `normalisieren` wird **gleich oft** aufgerufen und kostet 0.003 Sekunden. Wer ohne Profiling optimiert hätte, wäre gut über `normalisieren` gestolpert - die Funktion sieht teuer aus, `strip` und `lower` auf 3000 Strings. Sie ist es aber nicht.

Vorgehen bei Performance-Problemen

1. Reproduzierbaren Fall mit realistischen Daten bauen.
2. Mit `cProfile` den Hotspot suchen, nicht raten.
3. **Nur den Hotspot** ändern, alles andere in Ruhe lassen.
4. Mit `timeit` nachmessen und das Ergebnis mit `assert` auf Gleichheit prüfen.
5. Vorher- und Nachher-Zahl notieren - das ist der Nachweis für D1A.

Warum Messen und nicht Auswendiglernen

Eine oft zitierte Regel lautet: «Strings in einer Schleife mit += zusammenzuhängen ist quadratisch, nimm `join`.» Die Messung mit 20'000 Teilstücken:

```
s += t     0.00060 s
"".join()  0.00012 s     -> Faktor 5, nicht Faktor 1000
```

CPython optimiert genau dieses Muster intern, wenn der String nirgends sonst referenziert wird. `join` bleibt trotzdem die bessere Wahl - weil es kürzer und klarer ist und in anderen Python-Implementationen zuverlässig schneller. Aber die Begründung lautet **Lesbarkeit**, nicht Faktor 1000.

Genau das ist der Unterschied zwischen Niveau I und Niveau A: Auf Niveau I setzen Sie eine vorgegebene Massnahme um. Auf Niveau A prüfen Sie nach, ob sie in Ihrem Fall überhaupt etwas bringt - und sagen es ehrlich, wenn nicht.

Was Sie messen sollten

- **Realistische Datenmenge.** Mit 20 Datensätzen ist alles schnell genug.
- **Denselben Ablauf.** Kein Cache aus dem vorherigen Durchlauf, keine bereits geöffnete Datenbankverbindung nur in einer Variante.
- **Mehrere Wiederholungen**, und den kleinsten Wert nehmen.
- **Die Zahlen aufschreiben**, mit Datum und Datenmenge. Sonst ist die Messung in zwei Wochen wertlos.

M323-LU07, M323-D1I, M323-D1A