LU07.A06 - Unpure wird pure

Entfernen Sie den globalen Zustand aus einer kleinen Lagerverwaltung und trennen Sie Berechnung von Ein- und Ausgabe.

Repository: m323-lu07-a06-purerefactoring

Ausgangslage

```
lager = {}
protokoll = []


def einlagern(artikel, menge):
    global lager
    if artikel in lager:
        lager[artikel] += menge
    else:
        lager[artikel] = menge
    protokoll.append(f"+{menge} {artikel}")
    print(f"{artikel}: {lager[artikel]}")


def entnehmen(artikel, menge):
    global lager
    if lager.get(artikel, 0) < menge:
        print("zu wenig Bestand")
        return False
    lager[artikel] -= menge
    protokoll.append(f"-{menge} {artikel}")
    return True
```

```
einlagern("maus", 5)
einlagern("maus", 3)
entnehmen("maus", 2)      # True
entnehmen("maus", 99)     # False
print(lager)              # {'maus': 6}
```

Vorgehen

Führen Sie zuerst `pytest` aus. Sieben Tests sind **rot** - das ist Absicht. Die Tests sind hier die Spezifikation. Zwei davon lohnen sich genauer anzuschauen:

- `test_lauf_ist_wiederholbar` - zwei Durchläufe liefern verschiedene Ergebnisse, weil der

globale Zustand zwischen den Aufrufen hängen bleibt.
- `test_lauf_ausgabe` - schlägt fehl, sobald vorher ein anderer Test gelaufen ist. Genau das meint «Tests hängen voneinander ab».

Zu implementieren

```
mit_einlagerung(bestand, artikel, menge) -> dict
mit_entnahme(bestand, artikel, menge)    -> (dict, bool)
```

Beide Funktionen sind **pure**:

- Sie verändern `bestand` nicht, sondern geben einen neuen zurück.
- Sie geben nichts auf der Konsole aus.
- Derselbe Aufruf liefert immer dasselbe Ergebnis.

`lauf()` ist die Schale. Dort - und nur dort - leben Zustand, Protokoll und `print`. Am Schluss existieren die globalen Variablen `lager` und `protokoll` nicht mehr; `einlagern` und `entnehmen` in ihrer alten Form auch nicht.

Regeln

- `main_test.py` wird **nicht** verändert.
- Rückgabewerte und Ausgabe von `lauf()` bleiben exakt gleich.
- Kein `global` im fertigen Code.

Bewertung

Teil	Punkte
Tests (`main_test.py`)	14
pylint (`main.py`)	5

Leitfragen für die Reflexion

- Wie gibt eine Funktion gleichzeitig den neuen Bestand und die Information «hat geklappt / hat nicht geklappt» zurück?
- Wohin gehört das Protokoll - in den Kern oder in die Schale?
- Was wird an dieser Version einfacher zu testen als vorher?

M323-LU07, M323-D1I

© Kevin Maurizi

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

Permanent link:
https://wiki.bzz.ch/modul/m323/learningunits/lu07/aufgaben/purerefactoring?rev=1788953996

Last update: **2026/09/09 13:39**