LU07.L06 - Unpure wird pure

Funktionaler Kern

```
def mit_einlagerung(bestand, artikel, menge):
    """Neuer Bestand mit zusätzlicher Menge. Mutiert nichts."""
    return {**bestand, artikel: bestand.get(artikel, 0) + menge}


def mit_entnahme(bestand, artikel, menge):
    """(neuer Bestand, erfolgreich?) - bei zu wenig Bestand bleibt alles
gleich."""
    if bestand.get(artikel, 0) < menge:
        return bestand, False
    return {**bestand, artikel: bestand[artikel] - menge}, True
```

Schale

```
def lauf():
    bestand = {}
    protokoll = []

    bestand = mit_einlagerung(bestand, "maus", 5)
    protokoll.append("+5 maus")
    print(f"maus: {bestand['maus']}")

    bestand = mit_einlagerung(bestand, "maus", 3)
    protokoll.append("+3 maus")
    print(f"maus: {bestand['maus']}")

    bestand, ok = mit_entnahme(bestand, "maus", 2)
    if ok:
        protokoll.append("-2 maus")
    else:
        print("zu wenig Bestand")

    bestand, ok = mit_entnahme(bestand, "maus", 99)
    if not ok:
        print("zu wenig Bestand")

    return bestand, protokoll
```

```
lauf()  ->  ({'maus': 6}, ['+5 maus', '+3 maus', '-2 maus'])
```

Identisch zum Verhalten vorher.

Tests, die die Reinheit belegen

```
def test_kein_seiteneffekt():
    original = {"maus": 5}
    neu = mit_einlagerung(original, "maus", 3)
    assert neu == {"maus": 8}
    assert original == {"maus": 5}        # Original unverändert

def test_deterministisch():
    bestand = {"maus": 5}
    assert mit_entnahme(bestand, "maus", 2) == mit_entnahme(bestand, "maus",
2)

def test_zu_wenig_bestand():
    assert mit_entnahme({"maus": 1}, "maus", 99) == ({"maus": 1}, False)
```

Der Test `test_deterministisch` würde mit der alten Fassung scheitern: Dort liefert der zweite Aufruf einen anderen Bestand als der erste.

Antworten auf die Leitfragen

- **Zwei Rückgaben:** als Tuple `(bestand, ok)`. Alternativ eine frozen Dataclass `Ergebnis(bestand, erfolgreich, meldung)`, wenn mehr Information nötig wird.
- **Protokoll:** in die Schale. Es ist eine Aufzeichnung dessen, was passiert ist, nicht Teil der Berechnung. Wer es im Kern braucht, gibt es als weiteren Rückgabewert zurück, statt es zu mutieren.
- **Einfacher testbar:** Der Kern braucht keinen Startzustand und kein Aufräumen zwischen den Tests. Vorher musste `lager` und `protokoll` vor jedem Test zurückgesetzt werden - vergisst man das, hängen die Tests voneinander ab.

In Flask ist dieses Muster die Regel und nicht die Ausnahme: Die Route liest den Request und schreibt die Response (Schale), die Berechnung liegt in pure Funktionen, die ohne laufenden Server getestet werden können.

M323-LU07, M323-D1I