LU07.L04 - Lange Funktion zerlegen

Schritt 0: Sicherheitsnetz

```
# main_test.py
import main

ERWARTET = (
    "Bestellungen (4)\n"
    "----------------\n"
    "Meier: Maus = 75.0 CHF\n"
    "Keller: Laptop = 1140.0 CHF\n"
    "Meier: Tastatur = 152.0 CHF\n"
    "Keller: Monitor = 199.5 CHF\n"
    "----------------\n"
    "Total: 1566.5 CHF"
)

def test_bericht_unveraendert():
    assert main.bericht(main.BESTELLUNGEN) == ERWARTET

def test_rabatt_erst_ab_100():
    knapp_darunter = [{"kunde": "x", "artikel": "a", "menge": 1, "preis":
99.0, "status": "offen"}]
    assert "a = 99.0 CHF" in main.bericht(knapp_darunter)

def test_rabatt_ab_100_inklusive():
    genau_100 = [{"kunde": "x", "artikel": "a", "menge": 1, "preis": 100.0,
"status": "offen"}]
    assert "a = 95.0 CHF" in main.bericht(genau_100)

def test_leere_liste():
    assert main.bericht([]) == (
        "Bestellungen (0)\n"
        "----------------\n"
        "----------------\n"
        "Total: 0.0 CHF"
    )
```

Ergebnis nach dem Refactoring

```
RABATT_AB_BETRAG = 100
RABATT_PROZENT = 5
STATUS_STORNIERT = "storniert"
```

```
def ist_gueltig(bestellung):
    return bestellung["status"] != STATUS_STORNIERT


def zeilenwert(bestellung):
    wert = bestellung["menge"] * bestellung["preis"]
    if wert < RABATT_AB_BETRAG:
        return wert
    return wert * (1 - RABATT_PROZENT / 100)


def kundenname(bestellung):
    return bestellung["kunde"].strip().lower().capitalize()


def zeile(bestellung):
    artikel = bestellung["artikel"]
    return f"{kundenname(bestellung)}: {artikel} =
{round(zeilenwert(bestellung), 2)} CHF"


def bericht(bestellungen):
    gueltige = [b for b in bestellungen if ist_gueltig(b)]
    # Startwert 0.0: sonst liefert sum() bei leerer Liste ein int
    total = round(sum((zeilenwert(b) for b in gueltige), 0.0), 2)
    kopf = f"Bestellungen ({len(gueltige)})"
    strich = "-" * len(kopf)
    return "\n".join([kopf, strich] + [zeile(b) for b in gueltige] +
[strich, f"Total: {total} CHF"])
```

Commit-Folge

```
refactor: Guard-Bedingung als ist_gueltig() extrahiert
refactor: Rabattgrenze und -satz als Konstanten benannt
refactor: Zeilenwert in zeilenwert() extrahiert
refactor: Namensaufbereitung in kundenname() extrahiert
refactor: Zeilenformatierung in zeile() extrahiert
refactor: Schleife durch Comprehension und sum() ersetzt
refactor: String-Verkettung durch f-Strings ersetzt
```

Die Falle in dieser Aufgabe

Der naheliegende Umbau

```
total = round(sum(zeilenwert(b) for b in gueltige), 2)
```

ist **falsch**. Bei einer leeren Liste liefert `sum()` den int `0`, das Original arbeitet mit `total = 0.0` und liefert den float `0.0`. Der Bericht endet dann mit `Total: 0 CHF` statt `Total: 0.0 CHF`. Richtig ist der Startwert:

```
total = round(sum((zeilenwert(b) for b in gueltige), 0.0), 2)
```

Der Test `test_leere_liste` deckt genau das auf. Ohne ihn wäre die Änderung unbemerkt durchgegangen und später als kosmetischer Bug in der Rechnungsansicht wieder aufgetaucht. **Das ist der Grund, warum das Sicherheitsnetz vor dem ersten Umbau steht.**

Worauf es ankommt

- `ist_gueltig`, `zeilenwert` und `kundenname` sind **pure** und einzeln testbar. Vorher war nichts davon prüfbar, ohne den ganzen Bericht zu erzeugen.
- Die Rabattregel steht an einer Stelle und heisst so, wie die Fachabteilung sie nennt.
- `bericht` liest sich jetzt wie eine Inhaltsangabe: filtern, summieren, formatieren.
- Die Tests `test_rabatt_erst_ab_100` und `test_rabatt_ab_100_inklusive` decken die Zeile ab, an der ein `>` statt `>=` unbemerkt bliebe.

Zweite Stolperstelle: `round()` zu früh anwenden. Wer in `zeilenwert` rundet, bekommt bei anderen Datensätzen Rundungsdifferenzen im Total. Runden gehört an den Schluss.

M323-LU07, M323-D1I