LU01.A11 - Spaghetticode entwirren

Bringen Sie eine unübersichtliche Funktion auf die drei Grundstrukturen der strukturierten Programmierung zurück – Sequenz, Selektion und Iteration – ohne ihr Verhalten zu verändern.

Ausgangslage

Die folgende Funktion berechnet den Umsatz aus einer Liste von Bestellungen. Sie ist korrekt, aber niemand liest sie gern: eine `while`-Schleife mit Abbruchflagge, ein von Hand hochgezählter Index, `continue`-Sprünge und drei Verschachtelungsebenen.

```
def umsatz(bestellungen):
    i = 0
    fertig = False
    total = 0
    while not fertig:
        if i >= len(bestellungen):
            fertig = True
        else:
            b = bestellungen[i]
            if b["status"] == "storniert":
                i = i + 1
                continue
            else:
                if b["menge"] <= 0:
                    i = i + 1
                    continue
                else:
                    total = total + b["menge"] * b["preis"]
                    i = i + 1
    return total
```

Testdaten

```
bestellungen = [
    {"artikel": "Maus",     "status": "offen",     "menge": 3,  "preis":
24.90},
    {"artikel": "Tastatur", "status": "storniert", "menge": 2,  "preis":
79.50},
    {"artikel": "Monitor",  "status": "offen",     "menge": 0,  "preis":
249.00},
    {"artikel": "Kabel",    "status": "geliefert", "menge": 10, "preis":
9.90},
```

```
    {"artikel": "Dock",     "status": "offen",     "menge": 1,  "preis":
189.00},
]
```

Anforderungen

1. **Analysieren:** Markieren Sie im Ausgangscode, wo Sequenz, Selektion und Iteration vorkommen. Welche Stellen entsprechen keiner der drei Grundstrukturen, sondern imitieren einen Sprung?
2. **Umbauen:** Schreiben Sie die Funktion neu. Vorgaben:
   - Die Schleife läuft direkt über die Elemente (`for bestellung in bestellungen`), ohne Index und ohne Abbruchflagge.
   - Kein `continue`, kein `break`.
   - Höchstens **eine** Verschachtelungsebene innerhalb der Schleife.
   - Die Bedingung, ob eine Bestellung überhaupt zählt, steckt in einer eigenen, sprechend benannten Funktion, die `True` oder `False` zurückgibt.
3. **Beweisen:** Rufen Sie alte und neue Fassung mit denselben Testdaten auf und zeigen Sie mit einem `assert`, dass beide dasselbe Ergebnis liefern. Prüfen Sie zusätzlich die leere Liste.
4. **Begründen:** Halten Sie in zwei bis drei Sätzen fest, welche Fehlerquellen im Ausgangscode durch den Umbau verschwunden sind.

Beispieloutput

```
Umsatz alt: 362.70
Umsatz neu: 362.70
Beide Fassungen liefern dasselbe Ergebnis.
```

Zusatzfragen

1. Der Ausgangscode zählt `i` an drei verschiedenen Stellen hoch. Was passiert, wenn man eine dieser Stellen vergisst? Probieren Sie es aus.
2. Das Theorem von Böhm und Jacopini besagt, dass jeder Algorithmus mit den drei Grundstrukturen auskommt. Wie stützt Ihr Umbau diese Aussage?
3. **Für Schnelle:** Formulieren Sie die Anforderung an `umsatz` deklarativ – als Satz über das Ergebnis, ohne Schleife und Zwischensumme.

Wichtig

Das Verhalten der Funktion darf sich nicht ändern – auch nicht in Randfällen.
Prüfen Sie mindestens: leere Liste, alle Bestellungen storniert, Menge 0, Menge negativ.

Abgabe

Geben Sie die Python-Datei mit beiden Fassungen und den Tests sowie Ihre schriftliche Begründung in Moodle ab.

M323-LU01, M323-D1I, M323-C1A