LU01.L11 - Spaghetticode entwirren

1. Analyse des Ausgangscodes

Zeile / Konstrukt	Grundstruktur	Bemerkung
`i = 0`, `fertig = False`, `total = 0`	Sequenz	Drei Zustandsvariablen, die von Hand gepflegt werden müssen.
`while not fertig`	Iteration	Endlosschleife mit Abbruchflagge statt echter Laufbedingung.
`if i >= len(…)`	Selektion	Wird nur gebraucht, weil die Schleife nicht über die Elemente läuft.
`if b[„status“] == …` / `if b[„menge“] ⇐ 0`	Selektion	Sachlich sinnvoll, aber unnötig verschachtelt.
`continue`	**keine der drei**	Imitiert einen Sprung an den Schleifenanfang – genau das, was die strukturierte Programmierung vermeiden will.
`i = i + 1` an drei Stellen	Sequenz	Klassische Fehlerquelle: Eine vergessene Stelle erzeugt eine Endlosschleife.
`fertig = True`	Sequenz	Flaggenvariable als Ersatz für eine Schleifenbedingung.

2. Musterlösung

```
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},
]


def umsatz_alt(bestellungen):
    """Ausgangsfassung - bleibt zum Vergleich stehen."""
    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


def ist_verrechenbar(bestellung):
    """Selektion: Zaehlt diese Bestellung zum Umsatz?"""
    return bestellung["status"] != "storniert" and bestellung["menge"] > 0


def umsatz(bestellungen):
    """Sequenz - Iteration - Selektion, je genau einmal."""
    total = 0
    for bestellung in bestellungen:
        if ist_verrechenbar(bestellung):
            total += bestellung["menge"] * bestellung["preis"]
    return total


if __name__ == '__main__':
    print(f"Umsatz alt: {umsatz_alt(bestellungen):.2f}")   # 362.70
    print(f"Umsatz neu: {umsatz(bestellungen):.2f}")       # 362.70

    assert umsatz(bestellungen) == umsatz_alt(bestellungen)
    assert umsatz([]) == umsatz_alt([]) == 0
    assert umsatz([{"status": "storniert", "menge": 5, "preis": 10.0}]) == 0
    assert umsatz([{"status": "offen", "menge": -2, "preis": 10.0}]) == 0
    print("Beide Fassungen liefern dasselbe Ergebnis.")
```

3. Was durch den Umbau verschwunden ist

- **Der Index `i`.** Damit auch jede Möglichkeit, ihn falsch zu initialisieren, an einer Stelle nicht hochzuzählen (Endlosschleife!) oder um eins danebenzuliegen.
- **Die Flagge `fertig`.** Die `for`-Schleife kennt ihr Ende selbst.
- **Die Sprünge.** Ohne `continue` liest sich der Ablauf von oben nach unten.
- **Zwei Verschachtelungsebenen.** Die beiden Bedingungen sind zu einer benannten Frage zusammengefasst, die man laut vorlesen kann: *„Wenn die Bestellung verrechenbar ist, zähle sie dazu.„*

Übrig bleiben genau die drei Grundstrukturen: Sequenz (Initialisierung und Rückgabe), Iteration (`for`),

Selektion (`if`).

4. Antworten auf die Zusatzfragen

Frage 1: Vergessenes Hochzählen

Entfernt man beispielsweise das `i = i + 1` im Storno-Zweig, bleibt die Schleife für immer bei derselben stornierten Bestellung stehen – eine **Endlosschleife**. Das Programm hängt ohne Fehlermeldung. In der neuen Fassung ist dieser Fehler gar nicht mehr möglich, weil die Iteration nicht mehr von Hand gesteuert wird.

Frage 2: Böhm und Jacopini

Der Ausgangscode brauchte scheinbar Sprünge (`continue`), um „diesen Fall zu überspringen“. Die neue Fassung zeigt: Dieselbe Wirkung entsteht durch **Selektion** – statt zu überspringen, wird nur im passenden Fall etwas getan. Das ist genau die Aussage des Theorems: Jeder berechenbare Algorithmus lässt sich allein mit Sequenz, Selektion und Iteration ausdrücken. Sprünge sind bequem, aber nie zwingend nötig.

Frage 3: Deklarative Formulierung

„Der Umsatz ist die Summe aus Menge mal Preis über alle nicht stornierten Bestellungen mit positiver Menge.„

In Python direkt so ausdrückbar:

```
def umsatz_deklarativ(bestellungen):
    return sum(b["menge"] * b["preis"] for b in bestellungen if
ist_verrechenbar(b))
```

Weder Zwischensumme noch Schleifenkörper sind noch sichtbar – der Code liest sich wie der Satz darüber. Die Bausteine dahinter (`map`, `filter`, `reduce`, Comprehensions) behandeln wir ausführlich in LU04.

© Kevin Maurizi

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

Permanent link:
https://wiki.bzz.ch/modul/m323/learningunits/lu01/loesungen/spaghetticode?rev=1787040636

Last update: **2026/08/18 10:10**