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 Funktion `umsatz_alt` in `referenz.py` 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.

Diese Datei ist der Massstab: Ihre Fassung muss für jede Eingabe exakt dasselbe liefern. **Verändern dürfen Sie sie nicht.**

referenz.py

referenz.py

```
"""
Die urspruengliche Fassung der Umsatzberechnung - NICHT VERAENDERN.

Sie ist korrekt, aber niemand liest sie gern: eine while-Schleife mit
Abbruchflagge, ein von Hand hochgezaehlter Index, continue-Spruenge und
drei
Verschachtelungsebenen.

Ihre Fassung in main.py muss fuer jede Eingabe dasselbe Ergebnis
liefern.
Diese Datei wird nicht gelintet und nicht bewertet.
"""

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):
    """Berechnet den Umsatz - so, wie man es besser nicht schreibt."""
    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


if __name__ == '__main__':
    print(f'Umsatz alt: {umsatz_alt(BESTELLUNGEN):.2f}')
```

Code-Vorlage

Die Namen und Signaturen in `main.py` sind fix – die Tests rufen genau diese auf.

main.py

main.py

```
"""
LU01.A11 - Spaghetticode entwirren

Bringen Sie die Umsatzberechnung aus referenz.py auf die drei
Grundstrukturen
der strukturierten Programmierung zurueck - Sequenz, Selektion und
Iteration -
ohne ihr Verhalten zu veraendern.
```

```
Verboten sind in dieser Datei: while, continue und break.
"""

from referenz import BESTELLUNGEN


def ist_verrechenbar(bestellung):
    """
    Entscheidet, ob eine Bestellung zum Umsatz zaehlt.

    Eine Bestellung zaehlt, wenn sie nicht storniert ist und
    die Menge groesser als 0 ist.

    :param bestellung: dict mit artikel, status, menge, preis
    :return: True oder False
    """
    # TODO Beide Bedingungen zu einer Aussage zusammenfassen und
zurueckgeben


def umsatz(bestellungen):
    """
    Summiert Menge mal Preis ueber alle verrechenbaren Bestellungen.

    Verwenden Sie eine for-Schleife direkt ueber die Elemente, ohne
Index und
    ohne Abbruchflagge, und hoechstens eine Verschachtelungsebene
darin.

    :param bestellungen: Liste von Bestellungen
    :return: Umsatz als Zahl, bei leerer Liste 0
    """
    # TODO Sequenz - Iteration - Selektion, je genau einmal


def main():
    """Gibt den Umsatz der Beispielbestellungen aus."""
    print(f'Umsatz neu: {umsatz(BESTELLUNGEN):.2f}')


if __name__ == '__main__':
    main()
```

Anforderungen

Implementieren Sie in `main.py` die beiden vorgegebenen Funktionen. Vorgaben:

- Die Schleife läuft direkt über die Elemente (`for bestellung in bestellungen`), ohne

Index und ohne Abbruchflagge.
- Kein `continue`, kein `break`, kein `while`.
- Höchstens **eine** Verschachtelungsebene innerhalb der Schleife.
- Die Bedingung, ob eine Bestellung überhaupt zählt, steckt in `ist_verrechenbar` und gibt `True` oder `False` zurück.

Funktion	Rückgabe
`ist_verrechenbar(bestellung)`	`True`, wenn die Bestellung nicht storniert ist **und** die Menge grösser als 0 ist
`umsatz(bestellungen)`	Summe aus Menge mal Preis über alle verrechenbaren Bestellungen, bei leerer Liste `0`

Beispieloutput

```
Umsatz neu: 362.70
```

Schriftlicher Teil

Diese fünf Punkte beantworten Sie **direkt im `README.md`** Ihres Repositories, unterhalb der jeweiligen Frage. Zwei bis vier Sätze pro Punkt genügen.

1. **Grundstrukturen:** Wo kommen in `umsatz_alt` Sequenz, Selektion und Iteration vor? Welche Stellen entsprechen keiner der drei Grundstrukturen, sondern imitieren einen Sprung?
2. **Was der Umbau beseitigt:** Welche Fehlerquellen im Ausgangscode sind durch Ihren Umbau verschwunden?
3. **Vergessenes Hochzählen:** Der Ausgangscode zählt `i` an drei verschiedenen Stellen hoch. Was passiert, wenn man eine davon vergisst? Probieren Sie es in einer Kopie aus.
4. **Böhm und Jacopini:** Das Theorem besagt, dass jeder Algorithmus mit den drei Grundstrukturen auskommt. Wie stützt Ihr Umbau diese Aussage?
5. **Deklarativ formuliert (für Schnelle):** Formulieren Sie die Anforderung an `umsatz` als Satz über das Ergebnis, ohne Schleife und Zwischensumme.

Wichtig

Das Verhalten der Funktion darf sich nicht ändern – auch nicht in Randfällen.
Geprüft werden: leere Liste, alle Bestellungen storniert, Menge 0, Menge negativ.
Zwei weitere Tests lesen `main.py` als Syntaxbaum ein und prüfen, dass eine `for`-Schleife vorhanden ist und weder `while` noch `continue` oder `break` vorkommen.

Vorgehen

1. Akzeptiere das Classroom-Assignment
2. Klone dein persönliches Repository in die Entwicklungsumgebung
3. Analysiere `referenz.py`

4. Implementiere `ist_verrechenbar` und `umsatz` in `main.py`
5. Beantworte die fünf Punkte im `README.md`
6. Lokal prüfen mit `pytest` und `python _run_pylint.py`
7. Pushen – Code und Antworten liegen damit im selben Commit

Bewertung

Teil	Punkte
Unittests	11
Linting	5
Schriftlicher Teil (von Hand bewertet)	6
Total	**22**

Abgabe

Alles zusammen als Push in das persönliche GitHub-Repository: der Code in `main.py` und die Antworten im `README.md`. Es gibt keine separate Moodle-Abgabe.

⇒ *GitHub Repo für externe Besucher*

GitHub Repository https://github.com/templates-python/m323-lu01-a11-umsatz-refactoring

Lernende am BZZ müssen den Link zum Classroom-Assignment verwenden