LU01.A09 - Vom Skript zur Funktion

Bauen Sie ein Lagerverwaltungs-Skript, das ausschliesslich aus Prozeduren mit globalem Zustand besteht, in echte Funktionen um – so, dass jeder Aufruf bei gleicher Eingabe immer dasselbe Ergebnis liefert.

Ausgangslage

Das Skript in `ausgangslage.py` funktioniert – solange man es genau in dieser Reihenfolge aufruft. Es zeigt gleich mehrere typische Schwächen prozeduralen Codes: globalen Zustand, Berechnung und Ausgabe im selben Atemzug, und Prozeduren, die `None` zurückgeben.

ausgangslage.py

ausgangslage.py

```
"""
Die urspruengliche Fassung des Lagerskripts - NUR ZUM NACHLESEN.

Diese Datei wird nicht bewertet und muss nicht veraendert werden.
Sie zeigt, womit Sie starten: globale Variablen, Berechnung und Ausgabe
vermischt, und drei Prozeduren, die None zurueckgeben.
"""

lager = []
gesamtwert = 0


def artikel_hinzufuegen(name, menge, preis):
    """Haengt einen Artikel an die globale Liste und aktualisiert die
globale Summe."""
    global gesamtwert
    lager.append({'name': name, 'menge': menge, 'preis': preis})
    gesamtwert += menge * preis


def zeige_gesamtwert():
    """Druckt den globalen Gesamtwert."""
    print(f'Gesamtwert: {gesamtwert:.2f} CHF')


def zeige_teuersten():
    """Druckt den Namen des teuersten Artikels."""
    teuerster = None
```

```
    for artikel in lager:
        if teuerster is None or artikel['preis'] > teuerster['preis']:
            teuerster = artikel
    name = teuerster['name']
    print(f'Teuerster Artikel: {name}')


if __name__ == '__main__':
    artikel_hinzufuegen('Maus', 10, 24.90)
    artikel_hinzufuegen('Tastatur', 4, 79.50)
    artikel_hinzufuegen('Monitor', 2, 249.00)
    zeige_gesamtwert()
    zeige_teuersten()
```

Code-Vorlage

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

main.py

main.py

```
"""
LU01.A09 - Vom Skript zur Funktion

Bauen Sie das prozedurale Lagerskript in echte Funktionen um:
keine globalen Variablen, kein print in den berechnenden Funktionen,
jede Funktion gibt einen Wert zurueck, und die uebergebene Liste wird
nicht veraendert.

Die urspruengliche Fassung finden Sie in ausgangslage.py.
"""


def artikel_hinzufuegen(lager, name, menge, preis):
    """
    Gibt ein NEUES Lager zurueck, das zusaetzlich den angegebenen
Artikel enthaelt.
    Das uebergebene Lager darf dabei nicht veraendert werden.

    :param lager: Liste von Artikeln (dict mit name, menge, preis)
    :param name: Name des neuen Artikels
    :param menge: Stueckzahl des neuen Artikels
    :param preis: Stueckpreis des neuen Artikels
    :return: neue Liste mit allen bisherigen Artikeln plus dem neuen
```

```
    """
    # TODO Neue Liste erzeugen statt append verwenden


def gesamtwert(lager):
    """
    Berechnet den Gesamtwert aller Artikel im Lager (Menge mal Preis).

    :param lager: Liste von Artikeln
    :return: Gesamtwert als Zahl, bei leerem Lager 0
    """
    # TODO Summe berechnen und zurueckgeben


def teuerster_artikel(lager):
    """
    Sucht den Artikel mit dem hoechsten Stueckpreis.

    :param lager: Liste von Artikeln
    :return: der teuerste Artikel, bei leerem Lager None
    """
    # TODO Teuersten Artikel suchen und zurueckgeben


def formatiere_gesamtwert(lager):
    """
    Erzeugt den Ausgabetext zum Gesamtwert - und gibt ihn zurueck,
    statt ihn zu drucken.

    Format: 'Gesamtwert: 1065.00 CHF' (zwei Nachkommastellen)

    :param lager: Liste von Artikeln
    :return: Ausgabetext als String
    """
    # TODO Text zusammensetzen und zurueckgeben


def main():
    """Baut ein Lager auf und gibt die Auswertung aus."""
    lager = []
    lager = artikel_hinzufuegen(lager, 'Maus', 10, 24.90)
    lager = artikel_hinzufuegen(lager, 'Tastatur', 4, 79.50)
    lager = artikel_hinzufuegen(lager, 'Monitor', 2, 249.00)

    name = teuerster_artikel(lager)['name']
    print(formatiere_gesamtwert(lager))
    print(f'Teuerster Artikel: {name}')


if __name__ == '__main__':
```

```
    main()
```

Anforderungen

1. **Analyse:** Halten Sie fest, welche der drei `def`-Blöcke in `ausgangslage.py` Prozeduren sind und woran Sie das erkennen.
2. **Umbau:** Implementieren Sie in `main.py` die vier vorgegebenen Funktionen so, dass gilt:
   - Keine globalen Variablen – alle benötigten Daten kommen als Parameter herein.
   - Kein `print` innerhalb der berechnenden Funktionen. Ausgegeben wird ausschliesslich in `main()`.
   - Jede Funktion gibt einen Wert zurück.
   - `artikel_hinzufuegen` verändert die übergebene Liste **nicht**, sondern gibt eine **neue** Liste zurück.

Funktion	**Rückgabe**
`artikel_hinzufuegen(lager, name, menge, preis)`	eine **neue** Liste mit dem zusätzlichen Artikel
`gesamtwert(lager)`	Summe aus Menge mal Preis, bei leerem Lager `0`
`teuerster_artikel(lager)`	der Artikel mit dem höchsten Preis, bei leerem Lager `None`
`formatiere_gesamtwert(lager)`	Text im Format `Gesamtwert: 1065.00 CHF`

Beispieloutput

```
Gesamtwert: 1065.00 CHF
Teuerster Artikel: Monitor
```

Kontrollfragen

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

1. Warum kann man den Ausdruck `gesamtwert(lager)` im Kopf jederzeit durch seine Zahl ersetzen, den Aufruf `artikel_hinzufuegen('Maus', 10, 24.90)` aus der Ausgangslage aber nicht? Wie heisst diese Eigenschaft?
2. Was gibt `zeige_gesamtwert()` zurück? Probieren Sie `ergebnis = zeige_gesamtwert()` und geben Sie `ergebnis` aus.
3. Welche Ihrer neuen Funktionen liessen sich mit einem automatischen Test prüfen, welche der alten Prozeduren nicht? Begründen Sie.

Hinweis

Schleifen sind in dieser Aufgabe ausdrücklich erlaubt – innerhalb einer Funktion ist eine `for`-Schleife völlig in Ordnung. Es geht hier nicht um den schleifenfreien Stil, sondern um die Grenze zwischen **Funktion** und **Prozedur**: Liefert der Block einen Wert, oder *tut* er etwas?

Achtung beim Linting

`global` kostet in dieser Aufgabe Punkte (`W0603 global-statement` ist bewusst aktiviert) – genau darum geht es ja.

Vorgehen

1. Akzeptiere das Classroom-Assignment
2. Klone dein persönliches Repository in die Entwicklungsumgebung
3. Lies `ausgangslage.py`
4. Implementiere die vier Funktionen in `main.py`
5. Beantworte die drei Kontrollfragen 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
Kontrollfragen (von Hand bewertet)	4
Total	**20**

Abgabe

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

⇒ *GitHub Repo für externe Besucher*

GitHub Repository https://github.com/templates-python/m323-lu01-a09-lager-funktional

Lernende am BZZ müssen den Link zum Classroom-Assignment verwenden

© Kevin Maurizi

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

Permanent link:
https://wiki.bzz.ch/modul/m323/learningunits/lu01/aufgaben/prozedurzufunktion

Last update: **2026/08/18 11:20**