

Übungsblatt zu Vorlesung 4

Aufgaben mit `map()`, `filter()`, `reduce()`, Decorators, Currying

Aufgabe 1 — Textanalyse mit `filter()`

`filter()` nimmt eine Funktion und ein iterierbares Objekt entgegen.
Es werden nur diejenigen Elemente behalten, für die die Funktion `True` zurückgibt.

Gegeben:

```
satz = "die schnelle braune katze springt ueber den faulen hund"
```

Aufgabenstellung:

1. Teile den Satz in eine Wortliste auf (Tipp: `.split()`).
2. Verwende `filter()` und `lambda`, um nur Wörter mit mehr als 5 Buchstaben herauszufiltern.
3. Verwende `filter()` und `lambda`, um nur Wörter herauszufiltern, die mit einem Vokal beginnen (a, e, i, o, u).
4. Kombiniere beide Bedingungen in einem einzigen `filter()`-Aufruf:
Wörter mit mehr als 5 Buchstaben **UND** Vokal am Anfang.

Beispielausgabe:

```
lange_woerter -> ['schnelle', 'braune', 'springt', 'faulen']  
vokal_woerter -> ['ueber']  
beide         -> []
```

Zusatzaufgabe: Kombiniere `map()` und `filter()`: Wandle zuerst alle Wörter mit `map()` in Großbuchstaben um, filtere dann nach `Länge > 5`.

Aufgabe 2 — Satzbuilder und Berechnungen mit `reduce()`

`reduce(f, liste)` wendet `f` sukzessive auf das bisherige Ergebnis und das nächste Element an: `reduce(f, [x1, x2, x3]) = f(f(x1, x2), x3)`.
Importiere `reduce` aus `functools`.

Gegeben:

```
from functools import reduce

satzteile = ['Python', 'ist', 'eine', 'tolle', 'Sprache']
zahlen = [4, 2, 7, 1, 9, 3, 5]
```

Aufgabenstellung:

1. Verwende `reduce()`, um die Satzteile mit einem Leerzeichen getrennt zu einem einzigen String zusammenzuführen.
2. Verwende `reduce()`, um das Maximum der Zahlen-Liste zu finden – ohne `max()` zu verwenden.
3. Berechne $7!$ (Fakultät von 7) mit `reduce()` auf einer Liste, die du mit `range(1, 8)` erzeugst.

Beispielausgabe:

```
satz      -> 'Python ist eine tolle Sprache'
maximum   -> 9
7!        -> 5040
```

Hinweise:

- `reduce` kann auch einen Startwert als drittes Argument bekommen: `reduce(f, liste, startwert)`.
- Für das Maximum: `lambda a, b: a if a > b else b`.

Aufgabe 3 — `map`, `filter`, `reduce`

Gegeben sei die Liste `zahlen = [3, 7, 2, 8, 5, 10, 1, 6]`.

1. Verwendet `map` und ein `lambda`, um eine neue Liste mit den **Quadraten** aller Zahlen zu erzeugen.
2. Verwendet `filter` und ein `lambda`, um eine Liste mit nur den **geraden** Zahlen zu erhalten.
3. Verwendet `reduce` (aus `functools`!), um die **Summe** aller Zahlen zu berechnen.
4. Kombiniert alle drei: Berechnet die **Summe der Quadrate aller geraden Zahlen**.
Hinweis: `map` und `filter` geben *Iteratoren* zurück – um sie als Liste zu sehen, müsst ihr `list(...)` aussenrum schreiben.

Aufgabe 4 - Rekursion

Bei einem Lieferservice soll ein neues Sortiersystem implementiert werden. Dabei darf ein einzelnes Paket maximal 10 Kilo wiegen. Wenn es mehr als 10 Kilo wiegt, wird es in zwei kleinere Pakete aufgeteilt. Der Prozess wird so lange wiederholt, bis alle Pakete 10 Kilo oder weniger wiegen. Dabei soll gezählt werden, wie viele Pakete verwendet werden.

Aufgabenstellung:

- Implementiere rekursiv das System mit den o.g. Voraussetzungen.
- Überlege, welche alternative Herangehensweise man benutzen könnte, um immer die minimale Anzahl an Paketen versenden zu können.

Aufgabe 5 — Dekoratoren: Der Türsteher

Stell dir vor, du arbeitest an einer App und willst bestimmte Funktionen nur dann ausführen lassen, wenn der Nutzer eingeloggt ist. Du könntest das in jede Funktion einzeln einbauen – oder du schreibst einmal einen **Dekorator**, der das für dich erledigt.

Gegeben:

```
# Simulierter Login-Status
eingeloggt = True

def zeige_profil(name):
    print(f"Profil von {name} wird angezeigt.")

def kaufe_produkt(produkt, preis):
    print(f"Kauf bestaetigt: {produkt} fuer {preis} EUR")
```

Aufgabenstellung:

- Schreibe einen Dekorator `login_required`, der die dekorierte Funktion nur dann ausführt, wenn die Variable `eingeloggt` den Wert `True` hat. Andernfalls soll nur "Zugriff verweigert. Bitte einloggen." ausgegeben werden.
- Wende den Dekorator auf `zeige_profil` und `kaufe_produkt` an.

Teste beide Fälle: `eingeloggt = True` und `eingeloggt = False`.

Gerüst:

```
def login_required(func):
    def wrapper(*args, **kwargs):
        ### Hier kommt euer Code hin ###
    return wrapper

@login_required
def zeige_profil(name):
    print(f"Profil von {name} wird angezeigt.")

@login_required
def kaufe_produkt(produkt, preis):
    print(f"Kauf bestaetigt: {produkt} fuer {preis} EUR")
```

Erwartete Ausgabe (eingeloggt = True):

Profil von Anna wird angezeigt.
Kauf bestaetigt: Laptop fuer 999 EUR

Erwartete Ausgabe (eingeloggt = False):

Zugriff verweigert. Bitte einloggen.
Zugriff verweigert. Bitte einloggen.

Hinweise:

- Der Dekorator hat Zugriff auf globale Variablen wie `eingeloggt`.
- `*args` und `**kwargs` sorgen dafür, dass `wrapper` mit beliebig vielen Argumenten funktioniert.

Zusatzaufgabe: Mache `eingeloggt` zu einem Parameter des Dekorators, sodass du schreiben kannst: `@login_required(rolle='admin')`.
Der Dekorator soll dann auch die Rolle des Nutzers prüfen.

Aufgabe 6 – Currying

Verwende Currying, um schrittweise Tage in Sekunden umzurechnen.

Vorgehensweise: Schreibe für jeden Schritt (Tag zu Stunden, Stunden zu Minuten, usw.) eine eigene pure function und führe sie am Ende in einer Funktion zusammen, welche übergebene Funktionen nacheinander ausführt.