LU04d - Die map-Funktion in Python

Die map-Funktion ist eine eingebaute Funktion in Python, die verwendet wird, um eine bestimmte Funktion auf alle Elemente eines Iterables (z. B. eine Liste) anzuwenden. Die Syntax der `map`-Funktion ist wie folgt:

```
result = map(function, iterable)
```

- **function**: Die Funktion, die auf jedes Element des Iterables angewendet werden soll.
- **iterable**: Das Iterable, auf das die Funktion angewendet werden soll.

Das Ergebnis der `map`-Funktion ist ein neues Iterable, das die Ergebnisse der Funktion enthält.

Beispiel

Ein Beispiel für die Verwendung der map-Funktion ist die Quadratur aller Elemente einer Liste:

```
numbers = [1, 2, 3, 4]
squares = map(lambda x: x**2, numbers)
print(list(squares))  # Output: [1, 4, 9, 16]
```

Oder ohne die Verwendung einer Lambda-Funktion.

```
def square(number):
    return number ** 2

squares = map(square, [1, 2, 3, 4])
print(list(squares))  # Output: [1, 4, 9, 16]
```

Vergleich mit List Comprehensions

Die map-Funktion hat viele Gemeinsamkeiten mit List Comprehensions, aber es gibt auch Unterschiede.

- **Gemeinsamkeiten**:
  1. Beide können verwendet werden, um eine Funktion auf jedes Element eines Iterables anzuwenden.
  2. Beide erzeugen ein neues Iterable mit den transformierten Werten.
- **Unterschiede**:
  1. Die map-Funktion gibt ein Map-Objekt zurück, das in eine Liste konvertiert werden muss, während List Comprehensions direkt eine Liste zurückgeben.
  2. List Comprehensions können auch Bedingungen enthalten, um Elemente zu filtern. Mit map muss dazu `filter` verwendet werden.
  3. Die Syntax unterscheidet sich: List Comprehensions verwenden eine kompakte eckige

Klammer-Syntax, während `map` die Funktion und das Iterable als Parameter nimmt.

Beispiel mit List Comprehension

Das obige Beispiel mit der `map`-Funktion könnte auch mit einer List Comprehension geschrieben werden:

```
numbers = [1, 2, 3, 4]
squares = [x**2 for x in numbers]
print(squares)  # Output: [1, 4, 9, 16]
```

In vielen Fällen sind List Comprehensions die prägnantere und pythonischere Lösung. Sie bieten auch mehr Flexibilität, da sie auch Bedingungen zum Filtern enthalten können.

M323-LU04, M323-CG4, M323-CF4, M323-CE4