List Comprehensions

Ein mächtiges Werkzeug zum Erzeugen von Listen

Einführung

Die Listen-Abstraktion, eigentlich auch im Deutschen besser als "List Comprehension" bekannt, ist eine elegante Methode, Mengen in Python zu definieren oder zu erzeugen.

Die List-Comprehension kommt der mathematischen Notation von Mengen sehr nahe.

In Mathematik definiert man die Quadratzahlen der natürlichen Zahlen beispielsweise als $\{ x^2 \mid x \in \mathbb{N} \}$

Wozu?

List Comprehensions werden zum Erstellen von Listen, zum Filtern bestehender Listen und zum Mapping von Elementen in Listen auf Funktionen genutzt.

All das wäre aber auch ohne List Comprehensions möglich.

Grundsätzlich: eine List-Comprehension erzeugt immer genau eine neue Liste!

Listen erstellen mit List Comprehensions

Schema:

B = [Ausdruck for Element in Sequenz]

Liste = [x for x in [1,2,3,4]]

Menge aller x, die Element der Menge 1,2,3,4 sind

Liste = [(x, x**2) for x in range(1, 11)]

Menge aller Tupel (x, x^2)

Beispiel: Filtern einer bestehenden Liste

Filtere alle Zahlen aus Liste numbers, die kleiner sind als 4. Erstelle dazu eine neue Liste:

```
numbers = [1, 29, 3, 22, 4, 89]
small_numbers = []
for zahl in numbers:
    if zahl < 4:
        small_numbers.append(zahl)
```

als List Comprehension

```
small_numbers = [zahl for zahl in numbers if zahl < 4]
```

Mapping

Das Anwenden einer Funktion auf Elemente einer Sequenz nennt man Mapping.

[schreibe_file(a) for a in log_list]

Für jeden Eintrag a in log_list wird die (fiktive) Funktion schreibe_file(a) aufgerufen.

Eine Zuweisung zu einer neuen Variable ist hier nicht zwingend nötig, wenn uns der jeweilige Rückgabewert von schreibe_file nicht interessiert.

Bedingungen in der List Comprehension

Um Listen zu Filtern, kann man Bedingungen in der List Comprehension nutzen.

Liste = [Ausdruck for element in sequenz if Bedingung]

Liste = [x for x in [1,2,3,4] if x % 2 == 0]

Menge aller x, die Element der Menge 1,2,3,4 und durch 2 teilbar sind.

Ternärer Operator

In Python existiert wie in vielen anderen Sprachen der ternäre Operator

```
a = 3
b = 1 if a > 0 else 2
```

diesen Operator kann man auch in der List Comprehension nutzen:

```
A = [True if a % 2 == 0 else False for a in range(0, 100)]
```

Mehrere For-loops

Es ist möglich, auch mehr als einen for-Loop in der List-Comprehension zu nutzen.

C = [(a, b) for a in [1,2] for b in [2,4] if a > 0 and b > 0]