LU04c - List Comprehensions

List Comprehensions sind eine kompakte Schreibweise, um aus vorhandenen Daten neue Listen zu erzeugen. Man kann damit Elemente **transformieren** (z. B. quadrieren, in Grossbuchstaben umwandeln) und optional **filtern** – in einer einzigen, gut lesbaren Zeile. Die Schreibweise erinnert an mathematische Mengen-Schreibweise.

Hauptverwendungszweck von List Comprehensions

List Comprehensions werden in Python hauptsächlich für zwei Dinge verwendet:

- **Transformation**: Jedes Element eines Iterables wird in eine neue Form gebracht (z. B. x → `x**2` oder „`audi`“ → „`AUDI`“)
- **Filterung (optional)**: Es werden nur diejenigen Elemente übernommen, für die eine Bedingung erfüllt ist.

Typisch ist eine Kombination aus beidem:

> „Nimm alle Elemente aus einer bestehenden Datenquelle, filtere sie nach einer Bedingung und wandle sie in eine neue Form um.“

Die Syntax

Die allgemeine Form einer List Comprehension ist:

```
newlist = [expression for item in iterable if condition]
```

- **expression**: Der Ausdruck, dessen Wert in die neue Liste aufgenommen wird (oft eine Transformation von `item`).
- **item**: Der Name für das aktuelle Element in der Schleife.
- **iterable**: Die Datenquelle, über die iteriert wird (z. B. Liste, Tupel, range, String, ...).
- **condition** (optional): Eine Bedingung, die `True` oder `False` ergibt. Nur wenn sie `True` ist, wird `expression` in die neue Liste aufgenommen.

Ohne Filter-Bedingung sieht die einfachste Form so aus:

```
newlist = [expression for item in iterable]
```

Das ist äquivalent zu:

```
newlist = []
for item in iterable:
    # optional: if condition:
    newlist.append(expression)
```

Teile einer List Comprehension

Expression

Die **Expression** ist der Teil, der tatsächlich in der neuen Liste landet. Sie kann:

- einfach das aktuelle Element sein (`item`)
- oder eine beliebige Berechnung / Funktionsanwendung auf `item`.

1. **Beispiel mathematische Operation**:

```
# Expression = x * 2
doubled = [x * 2 for x in range(5)]
# Ergebnis: [0, 2, 4, 6, 8]
```

1. **Beispiel mit Funktionsaufruf**:

```
cars = ['audi', 'bmw', 'subaru', 'toyota']
newlist = [car.upper() for car in cars]
print(newlist)  # ['AUDI', 'BMW', 'SUBARU', 'TOYOTA']
```

Item

item ist der Name für das aktuelle Element der Iteration. Er wird im `for`-Teil definiert: `for item in iterable`.

1. **Beispiel**:

```
cars = ['audi', 'bmw', 'subaru', 'toyota']
newlist = [item.upper() for item in cars]
print(newlist)
```

Hier ist `item` nacheinander 'audi', 'bmw', 'subaru', 'toyota'.

Iterable

Das **Iterable** kann ein beliebiges iterierbares Objekt sein, z. B. eine Liste, ein Tupel, ein Set, ein String oder das Ergebnis von `range()`.

1. **Beispiel mit einer Liste**:

```
squared = [x**2 for x in [1, 2, 3, 4]]
# Ergebnis: [1, 4, 9, 16]
```

```
numbers = [1, 2, 3, 4]
squared = [x**2 for x in numbers]
# Ergebnis: [1, 4, 9, 16]
```

1. **Beispiel mit einem Tupel**:

```
doubled = [x * 2 for x in (1, 2, 3, 4)]
# Ergebnis: [2, 4, 6, 8]
```

1. **Beispiel mit range()**:

```
newlist = [x for x in range(10)]
# Ergebnis: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Condition

Die **Condition** ist ein optionaler Filter, der entscheidet, ob ein Element in die neue Liste aufgenommen wird. Nur wenn die Bedingung `True` ergibt, wird `expression` übernommen.

1. **Beispiel mit Filter**:

```
cars = ['audi', 'vw', 'seat', 'skoda', 'tesla']
newlist = [car for car in cars if car != 'tesla']
# newlist = ['audi', 'vw', 'seat', 'skoda']
```

Die Bedingung `car != 'tesla'` liefert `True` für alle Elemente ausser `'tesla'`, daher enthält die neue Liste alle Autos ausser Tesla.

Die Bedingung ist optional und kann weggelassen werden:

1. **Ohne Filter**:

```
newlist = [x for x in cars]
# Kopie der ursprünglichen Liste
```

Weitere Beispiele

1. **Mit einem Iterable eines anderen Typs (String)**:

```
letters = [letter for letter in 'Hallo Welt']
# Ergebnis: ['H', 'a', 'l', 'l', 'o', ' ', 'W', 'e', 'l', 't']
```

1. **Nested List Comprehensions (verschachtelt)**:

```python
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened = [num for row in matrix for num in row]
# Ergebnis: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Comprehensions sind ein mächtiges Werkzeug in Python, um vorhandene Daten in einer einzigen, klaren Zeile zu **transformieren** und (falls nötig) zu **filtern**. Richtig eingesetzt machen sie Code kürzer und oft besser lesbar.

M323-LU04, M323-B1B

© Kevin Maurizi

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

Permanent link:
https://wiki.bzz.ch/modul/m323/learningunits/lu04/listcomprehensions

Last update: **2026/08/18 10:24**